НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

На правах рукопису
УДК 004.93'1

До захисту допущено
В. о. завідувача кафедри ММСА

О.Л.Тимощук

«___» _____ 2019 р.

Магістерська дисертація

на здобуття ступеня магістра за спеціальністю 122 Комп'ютерні науки
на тему: «Застосування глибинних нейронних мереж для розпізнавання образів при навчанні в автосимуляторах»

Виконав:
студент ІІ курсу, групи КА-83 мп
Чапалюк Максим Володимирович _____

Керівник:
доцент кафедри ММСА,
к.ф.-м.н., доцент Шубенкова І.А. _____

Рецензент:
професор кафедри АУТС ФІОТ
д.т.н., доцент Корнієнко Б.Я _____

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів
без відповідних посилань

Студент _____

Київ
2019

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Рівень вищої освіти — другий (магістерський)
Спеціальність (спеціалізація) — 122 «Комп'ютерні науки» («Системи штучного інтелекту»)

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри ММСА
О. Л. Тимощук
«___» _____ 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту Чапалюку Максиму Володимировичу

1. Тема дисертації: «Застосування глибинних нейронних мереж для розпізнавання образів при навчанні в автосимуляторах», науковий керівник дисертації Шубенкова Ірина Анатоліївна, к.ф.-м.н., доцент, затверджені наказом по університету від « 8 » листопада 2019 р. № 3862-с

2. Термін подання студентом дисертації: 13 грудня 2019 р.

3. Об'єкт дослідження: глибинні нейронні мережі

4. Предмет дослідження: застосування глибинних нейронних мереж в задачах розпізнавання образів.

5. Перелік завдань, які потрібно розробити:
1) спроєктувати систему розпізнавання образів на дорозі;
2) вибрати параметри системи, які необхідно буде оптимізовувати;
3) спроєктувати окрему систему для підбору параметрів;
4) знайти оптимальні значення параметрів системи;
5) розробити консольний додаток для розпізнавання образів та перевірити точність його роботи;

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:

1). Презентація 15 слайдів

7. Дата видачі завдання: _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Пошук матеріалів, що підтверджують актувальність та новизну теми	05.09.2019 – 11.09.2019	
2.	Проєктування та розробка додатку Розпізнавання образів на дорозі (РО) з використанням CNN	12.09.2019 – 21.09.2019	
3.	Проєктування схеми консольного додатку РО	22.09.2019 – 05.10.2019	
4.	Вибір параметрів, які необхідно оптимізувати	06.10.2019 – 08.10.2019	
5.	Проєктування системи для підбору параметрів	09.10.2019 – 10.10.2019	
6.	Реалізація системи для підбору параметрів	11.10.2019 – 20.10.2019	
7.	Додаткова оптимізація параметрів CNN	21.10.2019 – 24.10.2019	
13.	Розробка консольного додатку на базі спроєктованої системи РО та знайдених параметрів	25.10.2019 – 16.11.2019	
14.	Розгляд консольного додатку як стартап-проєкту та проведення для нього маркетинг-аналізу	17.11.2019 – 18.11.2019	
15.	Оформлення звіту	19.11.2019 – 25.11.2019	

Студент М.В. Чапалюк

Науковий керівник дисертації І.А. Шубенкова

РЕФЕРАТ

Магістерська дисертація виконана на 87 сторінках, містить 17 ілюстрацій, 22 таблиці та 1 додаток.

Дисертацію присвячено актуальній темі – системі розпізнавання образів на дорозі.

Мета дослідження – використовуючи глибинні нейронні мережі, розробити систему, яка буде розпізнавати дорожні знаки, дорожню розмітку та інші ТЗ.

Об’єкт дослідження – глибинні нейронні мережі.

Предмет дослідження – застосування глибинних нейронних мереж в задачах розпізнавання образів.

Методи дослідження – аналіз глибинних нейронних мереж, експериментальний вибір перної архітектури.

Наукова новизна – було розроблено програму, задача якої полягала у розпізнаванні дорожніх знаків, дорожньої розмітки та інших ТЗ.

Вдосконалити продукт можливо шляхом створення власного автопілота на базі розробленої програми

РОЗПІЗНАВАННЯ ОБРАЗІВ, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, ДОРОЖНІ ЗНАКИ, ДОРОЖНЯ РОЗМІТКА, ТРАНСПОРТНІ ЗАСОБИ

ABSTRACT

Master's thesis performed at 87 pages, contains 17 illustrations, 1 supplement.

The thesis is devoted to the actual topic – The system of pattern recognition on the road.

The aim – is to develop a system that will recognize road signs, road markings and other vehicles using deep neural networks.

The object of study – deep neural networks.

Purpose of the study – usage of deep neural networks in pattern recognition tasks.

Methods – analysis of deep neural networks, experimental choice of the architecture.

Scientific novelty – a program was developed, the task of which was to recognize road signs, road markings and other vehicles.

It is possible to improve the product by creating your own autopilot based on the developed program

IMAGE RECOGNITION, CONVERTING NEURAL NETWORK, ROAD SIGNS, ROAD MARKING, VEHICLES

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ANN — Artificial neural network, Штучна нейронна мережа.

CNN — Convolutional neural network, Згорткова нейронна мережа.

DNN — Deep neural network, Глибинна нейронна марежа.

ТЗ — Транскортні засоби.

ВСТУП

У сучасному автомобілебудуванні все частіше зустрічаються технічні системи, спрямовані на оптимізацію водіння автомобіля, що включають в себе також системи автоматичного розпізнавання дорожніх знаків, інших учасників руху та дороги з власною розміткою. Це спрощує завдання водія, дозволяє йому краще зосередитися на процесі водіння, підвищує безпеку всіх учасників дорожнього руху.

Одним з варіантів системи ідентифікації можуть бути радіометричні «маяки», здатні повідомити радіо-модулю автомобіля про те, що він увійшов в зону дії даного знаку чи дорожньої розмітки. Однак, система безпосереднього розпізнавання знаків та розмітки з зображень має переваги в плані надійності і можливості її застосування до існуючої інфраструктури дорожніх знаків та дорожньої розмітки. Крім того, данна система може розпізнавати й інші автомобілі на дорозі, що підвищує безпеку. При цьому

Розпізнавання дорожньої обстановки відбувається за наступним алгоритмом:

1. відеокамера охоплює область дороги, в якій можуть розташовуватися дорожні знаки, інші автомобілі та дорожню розмітку по ходу руху автомобіля;
2. дані передаються на вхід інформаційної системи, яка здійснює пошук дорожніх знаків і їх ідентифікацію, пошук автомобілів на дорозі, а також пошук дорожної розмітки.

Для вирішення задачі розпізнавання дорожніх знаків застосовують різні методи, що спираються на такі підходи, як порівняння з шаблонами, алгебраїчні моменти, лінії однакової інтенсивності, еластичні (деформуються) еталони порівняння, використання нейронних мереж і т.д.

В данній роботі будуть використовуватись саме глибинні нейронні мережі, зокрема згорткові нейронні мережі.

ПОСТАНОКА ЗАДАЧІ

Метою даної роботи є розробка системи для знаходження та розпізнавання образів на дорозі, а саме знаходження та розпізнавання дорожніх знаків, дорожної розмітки та інших автомобілів.

Основні задачі, що підлягають розв‘язання:

а)опис моделі для розпізнавання образів за допомогою згорткових нейронних мереж;

б)проведення огляду існуючих підходів до розпізнавання образів на зображенні;

в)проведення порівняльного аналізу існуючих методів розпізнавання образів на зображеннях;

г)вибір відповідного методу;

д)розробка програмного забезпечення;

е)тестування програмного рішення;

ж)аналіз та інтерпретація отриманих результатів.

1 ГЛИБИННІ НЕЙРОННІ МЕРЕЖІ

Глибоке навчання є частиною більш широкого сімейства методів машинного навчання на основі вивчення уявлень даних. Спостереження, наприклад зображення, можуть бути представлені в багатьох формах, таких як вектор значень інтенсивності на піксель, або в більш абстрактній формі у вигляді набору ребер, областей певної форми і т. д. Деякі уявлення краще за інших, що дозволяє спростити задачу навчання (наприклад, розпізнавання осіб або розпізнавання виразу особи). Одна з переваг глибокого навчання полягає в автоматичному виборі ознак.

Дослідження в цій області націлені на вибір найкращого представлення об'єктів. Деякі з уявлень використовують досягнення в області нейробіології, зокрема, ґрунтуються на інтерпретаціях обробки інформації та комунікаційних зв'язків в нервовій системі, таких як нейрони кодування, яке намагається визначити взаємозв'язок між різними стимулами і пов'язані з ними нервові реакції в головному мозку.

Різні спеціальні нейромережеві архітектури, такі як глибокі нейронні мережі (DNN), згорткові нейронні мережі (CNN), були успішно застосовані в таких областях, як комп'ютерний зір, автоматичне розпізнавання мови, обробка природної мови, біоінформатика. Нещодавно з'явилася фундаментальна монографія про Deep Learning.

1.1 Фундаментальні концепції

Глибокі моделі засновані на розподілених уявленнях. Основним припущенням теорії розподілених представлень є те, що спостережувані дані

генеруються за допомогою взаємодії факторів, організованих в шари. глибоке навчання передбачає, що ці шари факторів відповідають рівням абстракції або композиції. Змінюючи кількість шарів і розмірів шарів, ми можемо досягти різних рівнів абстракції.

Глибоке навчання використовує ідею ієрархічних пояснюють факторів, де більш високий рівень, більш абстрактні поняття, витягуються з нижніх рівнів. Ці архітектури часто будуються шар за шаром. Глибоке навчання допомагає розплутати ці абстракції і виявити, які функції можуть бути корисні для вивчення.

Важлива перевага глибоких моделей – це те, що для них були розвинені алгоритми навчання без супервайзера (вчителі).

1.2 Короткий огляд структури глибоких нейронних мереж

Глибока нейронна мережа (DNN) є штучною нейронною мережею (ANN) з декількома прихованими шарами між вхідним і вихідним шарами. Подібно звичайній ANN, DNN може моделювати складні нелінійні відносини. DNN архітектури, наприклад, для виявлення об'єкта і синтаксичного аналізу, генерують композиційні моделі, де об'єкт виражається у вигляді шаруватої композиції примітивів зображення.

DNN, як правило, виконані у вигляді мереж прямого поширення, але тут також дуже успішно застосовуються рекурентні нейронні мережі.

Згорткові нейронні мережі (CNN) використовуються в області комп'ютерного зору. CNN також були застосовані до акустичного моделювання для автоматичного розпізнавання мови (ASR), де вони були успішні в порівнянні з попередніми моделями (був період, коли в цій області домінували приховані

марковские моделі). Ми коротко розглянемо історію методів навчання DNN нижче.

1.3 Зворотне поширення помилки

DNN може бути навчена стандартним алгоритмом backpropagation. Згідно з різними джерелами, основи безперервного зворотного поширення були отримані в контексті теорії управління Генрі Дж. Келлі в 1960 р і Артуром Е. Брайсон в 1961 р, який застосував принципи динамічного програмування. У 1962 р Стюарт Дрейфус опублікував понад простий висновок, заснований тільки на ланцюговому правилі. У 1970 р Сеппо Ліннайнмаа опублікував загальний метод автоматичного диференціювання дискретних пов'язаних мереж за допомогою вкладених диференціюються (AD). Це відповідає сучасної версії методу backpropagation, який є ефективним, навіть коли мережі слабо пов'язані. В 1973 р Стюарт Дрейфус використовується алгоритм backpropagation для адаптації параметрів контролерів пропорційно градиентам помилок. У 1974 р Пол Уербос згадав про можливість застосування цього принципу в штучних нейронних мереж, і в 1982 р, він застосував метод AD Linnainmaa до нейронних мереж, який широко використовується в даний час. В 1986 р Девід Е. Румельхарт, Джеффрі Е. Хінтон і Рональд Дж. Вільямс показали, за допомогою комп'ютерних експериментів, що цей метод може генерувати корисні внутрішні уявлення вхідних даних в прихованих шарах нейронних мереж. У 1993 р Ерік А. Ван був першим, хто виграв міжнародний конкурс розпізнавання образів за допомогою backpropagation.

1.4 Труднощі з глибокими моделями

Як і в разі ANN, можуть виникнути труднощі з навчанням DNN, якщо вони навчаються примітивними наївними способами. Дві основні проблеми – перенавчання і великий час обчислень.

Домінуючий метод для навчання структур DNN – це корекція помилок навчання (наприклад, backpropagation з градієнтним спуском) завдяки простоті реалізації і тенденції сходитися на краще локальних оптимумів, ніж інші методи навчання. Проте, ці методи можуть бути обчислювально дорогими, особливо для DNN (тобто вимагати більшого часу навчання або пам'яті). Є багато параметрів навчання, які необхідно враховувати при навчанні DNN, такі як розмір (кількість шарів і кількість одиниць на шар), швидкість навчання та початкових ваг. Охоплення всього простору параметрів для оптимальних параметрів може виявитися неможливим через витрати часу і обчислювальних ресурсів. Було показано, що різні «Трюки», такі як використання міні-пакетування (обчислення градієнта на кількох навчальних прикладах відразу, а не на окремих прикладах), корисні для прискорення обчислень. Велика пропускна здатність обробки графічних процесорів виправила значні прискорення в навчанні в силу векторно-матричного характеру обчислень, які тут необхідні і добре виконуються графічними процесорами.

1.5 Згорткові нейронні мережі

CNN зручні для обробки візуальних та інших двовимірних даних. CNN складається з одного або більше згортальних шарів з повністю з'єднаними нейронами. У порівнянні з іншими глибокими архітектурою, згорткові нейронні

мережі показали чудові результати в обробці зображень і мовних додатках. Вони також можуть бути навчені за допомогою стандартного зворотного поширення. CNN легше навчаються, ніж інші регулярні, глибокі нейронної мережі і мають багато менше параметрів для оцінки, що робить їх дуже привабливими.

1.6 Розпізнавання образів

Звичайною оцінкою набору для класифікації зображень є набір даних бази даних MNIST. MNIST складається з рукописних цифр і включає в себе 60000 прикладів для навчання і 10000 тестових прикладів. Як і в разі TIMIT, його невеликий розмір дозволяє створювати кілька конфігурацій для тестування. Нині кращий результат на MNIST є коефіцієнт помилки 0,23%, досягнутий Ciresan і ін. в 2012 році

Одним із прикладів додатків DNN є навчання з поглибленим вивченням, що може дозволити автомобілю зробити повний огляд свого оточення. Іншим прикладом є технологія, відома як дісморфології особи, яка використовується для аналізу випадків вад розвитку людини та пов'язана з аналізом великою базою даних генетичних синдромів.

1.7 Висновки до розділу

В данному розділі було розглянуто загальну інформацію про глибинні нейронні мережі, іх структуру та випадки, в яких іх доручно використовувати. Плюсами таких мереж можна вважати навчання без учителя, а також розпізнавання більш глибоких, деколи неочевидних, закономірностей в данних.

В таких мережах за кожен рівень ознак відповідає свій шар і вони чудово себе показують в задачах розпізнавання образів на зображенні та в задачах розпізнавання голосу. Недоліками таких мереж є перенавчання і великий час обчислень.

Зрозуміло, що для вирішення задач, поставлених в межах дослідження данної дисертації доцільно використовувати саме CNN, адже вони чудово себе показують в задачах розпізнавання образів на зображеннях.

2 ТЕОРІЯ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

2.1 Згорткові нейронні мережі

CNN – спеціальна архітектура штучних нейронних мереж, запропонована Яном Лекуном і націлена на ефективне розпізнавання зображень, входить до складу технологій глибинного навчання (англ. deep learning). Використовує деякі особливості зорової кори, в якій були відкриті так звані прості клітини, що реагують на прямі лінії під різними кутами, і складні клітини, реакція яких пов'язана з активацією певного набору простих клітин. Таким чином, ідея CNN полягає в чергуванні згуткових шарів (англ. Convolution layers) і субдіскретізіруючіх шарів (англ. Subsampling layers або англ. Pooling layers, шарів підвибірки) (рисунок 2.1). Структура мережі - односпрямована (без зворотних зв'язків), принципово багатошарова. Для навчання використовуються стандартні методи, найчастіше метод зворотного поширення помилки. Функція активації нейронів (передавальна функція) - будь-яка, за вибором дослідника.

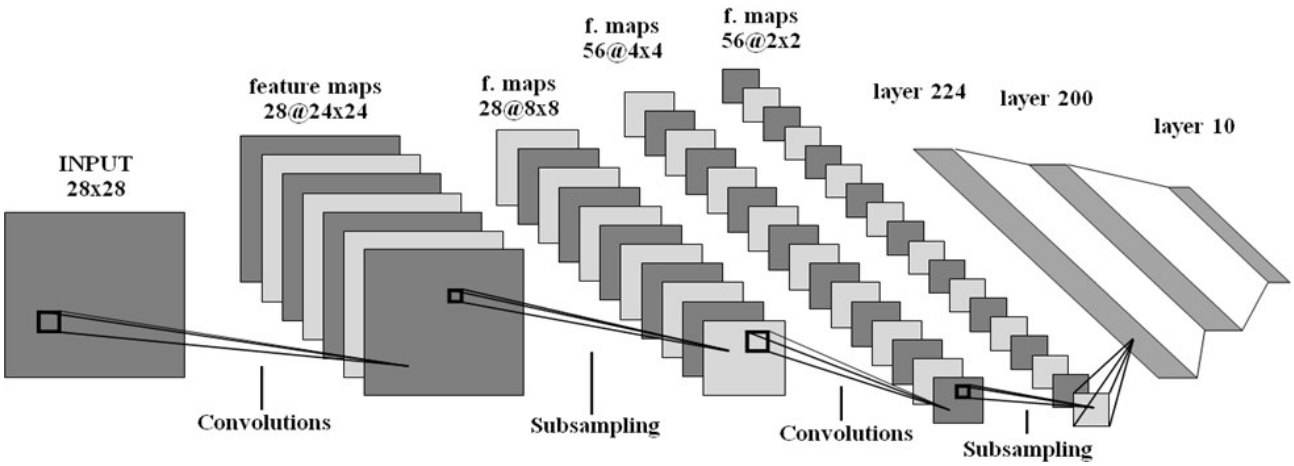

Рисунок 2.1 - Приклад архітектури CNN.

Назва архітектура мережі отримала через наявність операції згортки, суть якої в тому, що кожен фрагмент зображення множиться на матрицю (ядро) згортки поелементно, а результат підсумовується і записується в аналогічну позицію вихідного зображення.

2.2 Формальне визначення CNN

Штучний нейрон (як і природний) характеризується поточним станом і володіє групою синапсів - односпрямованих вхідних зв'язків, з'єднаних з виходами інших нейронів.

Також нейрон має аксон - вихідний зв'язок даного нейрона, з якого сигнал (збудження або гальмування) надходить на синапси наступних нейронів (рисунок 2.2).

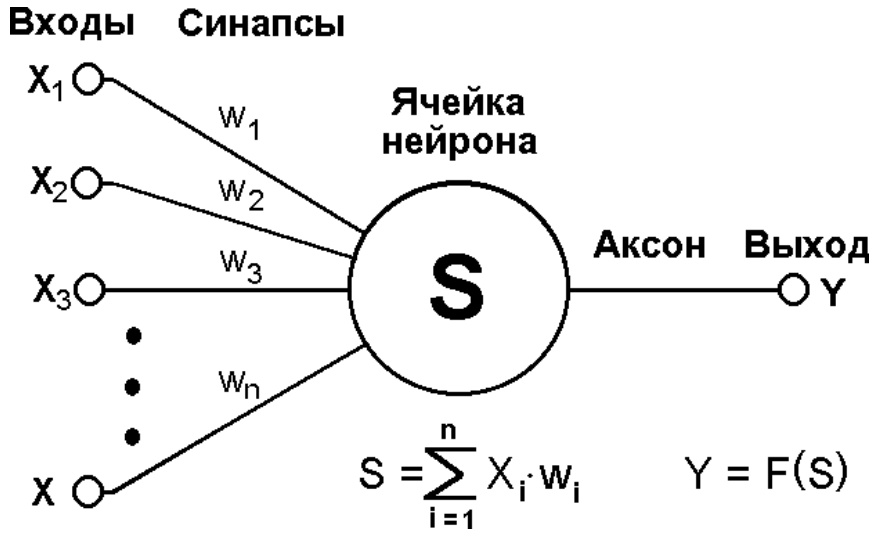

Рисунок 2.2 - Штучний нейрон

Кожен синапс характеризується величиною синаптичного зв'язку (її вагою wi).

Поточний стан нейрона визначаєтьс $S=\sum_{i=1}^{n} x_i * w_i$ сума його входів:

(2.1)

Вихід нейрона є функція його стану:

$$y = f(s) \tag{2.2}$$

Кожен шар даних в згуткової мережі являє собою тривимірний масив розміру h × w × d, де h і w - просторові виміри, а d - це характеристика або розмір каналу. Перший шар - це зображення з розмірами пікселів h × w і d квітів. Нейрони в верхніх шарах відповідають пікселям на зображенні, до якого вони підключені. Такі шари називаються полями сприйняття (receptive fields). Згуткові мережі будуються на трансляційної інваріантності. Їх базові компоненти (згортка, пулінг і активаційні функції) взаємодіють з регіонами шару введення і залежать від відносних просторових координат. Більш докладно про це буде розказано нижче.

Якщо записати вектор даних за адресою (i, j) в даному шарі як xij, а в наступному шарі - як yij, то висновок функції розраховує результат на наступному шарі можна записати так:

$$y_{ij} = f_{ks}\left(\{x_{si+\delta i, sj+\delta j}\}_{0 \le \delta i, \delta j \le k}\right) \tag{2.3}$$

де k - розмір ядра згортки, s - зсув ядра згортки (stride or subsampling factor), а fks - функція обумовлена типом шару: множення матриць для згортки, визначення середнього числа в матриці для пулінг за середнім значенням, просторовий

максимум (spatial max) для макспулінгу, розрахунок функції активації і так далі для інших типів шарів.

Така функціональна форма описується композицією з розміром ядра і його зміщенням котрі підпорядковуються правилу:

$$f_{ks} \circ g_{k`s`} = (f \circ g)_{k`+(k-1)s`,ss`} \tag{2.4}$$

У той час як загальна нейронна мережа обчислює загальну нелінійну функцію, мережа з тільки рівнями цієї форми обчислює нелінійний фільтр, який називається глибоким фільтром або повно сверточное мережею (fully convolutional network). FCN, природно, працює на вході будь-якого розміру і виробляє вихід відповідних (можливо, повторно обраних) просторових вимірів.

Матеріальна функція втрат, складена за допомогою FCN, визначає завдання. Якщо функція втрат є сумою по просторовим розмірами кінцевого шару, $\ell(x; \theta) = \Sigma_{ij} \ell'(x_{ij}; \theta)$ її градієнт буде сумою по градієнтам кожної з її просторових компонент. Таким чином, стохастичний градієнтний спуск на ℓ, обчислений на цілих зображеннях, буде таким же, як і стохастичний градієнтний спуск на ℓ', приймаючи все кінцеві рецептивні поля у вигляді міні-пакета.

Коли ці поля сприйняття накладаються один на одного, як пряме обчислення, так і зворотне поширення набагато ефективніше при обчисленні пошарово по всьому зображенню замість незалежного патча за патчем.

2.3 Шар згортки

Шар згортки (англ. convolutional layer) - це основний блок згуткової нейронної мережі. Шар згортки включає в себе для кожного каналу свій фільтр, ядро згортки якого обробляє попередній шар за фрагментами (підсумовуючи результати матричного твори для кожного фрагмента) (рисунок 2.3). Вагові коефіцієнти ядра згортки (невеликий матриці) невідомі і встановлюються в процесі навчання.

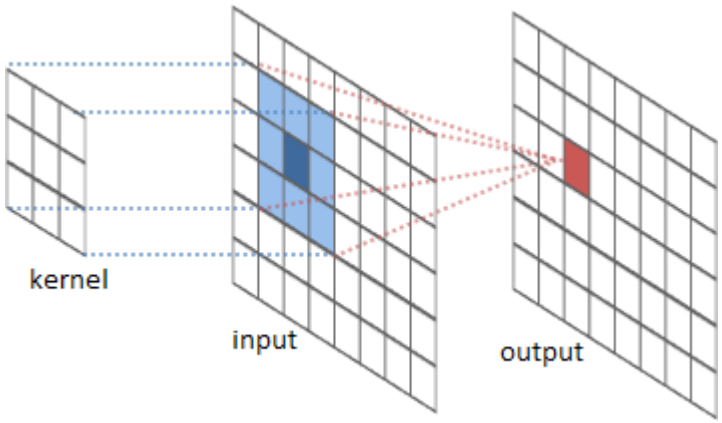

Рисунок 2.3 - Операція згортки

Особливістю згуткового шару є порівняно невелика кількість параметрів, яке встановлюється при навчанні. Так, наприклад, якщо вихідне зображення має розмірність 100x100 пікселів по трьом каналам (це значить 30000 вхідних нейронів), а згутковий шар використовує фільтри с ядром 3x3 пікселя з виходом на 6 каналів, тоді в процесі навчання визначається тільки 9 ваг ядра, однак за всіма сполученням каналів, тобто 3x3x3x6 = 162, в такому випадку даний шар вимагає знаходження тільки 162 параметрів, що істотно менше кількості шуканих параметрів повно нейронної мережі.

2.4 Шар ReLu

ReLU - це скорочення від англійського англ. rectified linear unit (ReLU), і означає блок лінійної ректифікації. По суті справи шар ReLU - ні що інше як функція активації після згуткового шару. Однак, для активації вибирається замість звичайних функцій типу гіпертангенса:

$$f(x)=\tanh(x), f(x)=\left|(\tanh(x))\right| \tag{2.5}$$

сигмоїди:

$$f(x) = (1 + e^{-x})^{-1} \tag{2.6}$$

або ненаситна функція:

$$f(x)=max(0,x) \tag{2.7}$$

Така функція показує хороші результати при навчанні нейронних мереж і відповідає за відсікання непотрібних деталей в каналі (при негативному виході). (Англ. Rectified linear unit (ReLU)).

2.5 Пулінг або шар субдіскретізації

Шар пулінг (інакше підвибірки, субдіскретізації) являє собою нелінійне ущільнення карти ознак, при цьому група пікселів (зазвичай розміру 2x2) ущільнюється до одного пікселя, проходячи нелінійне перетворення. Найбільш споживані при цьому функція максимуму (рисунок 1.4).

Перетворення зачіпають непересічні прямокутники або квадрати, кожен з яких скорочується в один піксель, при цьому вибирається піксель, що має максимальне значення. Операція пулінг дозволяє істотно зменшити просторовий обсяг зображення. Пулінг інтерпретується так. При виявленні деяких ознак на попередній операції згортки для подальшої обробки вже не потрібне настільки докладне зображення, і воно ущільнюється до менш докладного. До того ж фільтрація вже непотрібних деталей допомагає не перенавчатися. Шар пулінг, як правило, вставляється після шару згортки перед шаром наступної згортки.

Крім пулінгу з функцією максимуму можна використовувати і інші функції - наприклад, середнього значення або L2-нормування. Однак практика показала переваги саме пулінгу з функцією максимуму, який включається в типові системи.

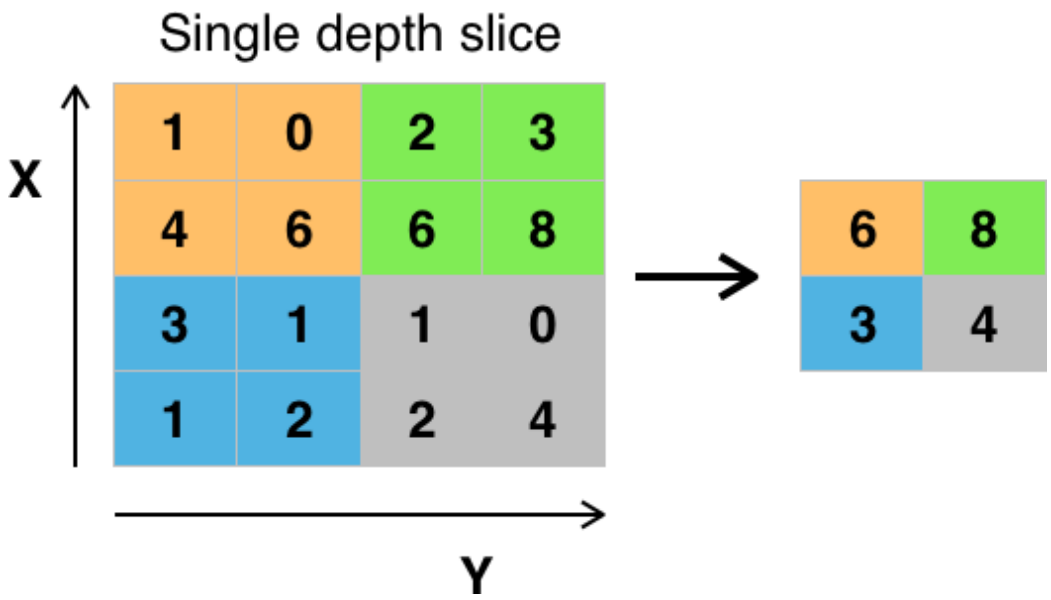

Рисунок 2.4 - Макспулінг

Крім завдань ущільнення зображення (що корисно для запобігання перенавчання), знаходять поширення також ідеї використання малих фільтрів або повна відмова від шарів пулінг в архітектурі мережі.

2.6 Оптимізатори

Оптимізатор організовує оптимізацію моделі, координуючи прямий і зворотний градієнти мережі для формування оновлень параметрів, які намагаються поліпшити функцію втрат. Обов'язки навчання розподіляються наступним чином: оптимізатор використовується для контролю оптимізації та генерації оновлень параметрів, а мережа - для отримання збитків і градієнтів.

Стохастичний градієнтний спуск (часто скорочений в SGD), також відомий як інкрементний градієнтний спуск, є стохастичною апроксимацією методу оптимізації спуску градієнта для мінімізації цільової функції, яка записується як сума

диференціюються. Іншими словами, SGD намагається знайти мінімуми або максимуми шляхом ітерації.

2.7 Розрахунок споживання пам'яті мережею

Сучасні графічні процесори мають ліміт пам'яті в 3, 4 або 6 Гб, в той час як кращі GPU розташовують 12 Гб. Є 3 основних джерела об'єктів займають пам'ять:

— з проміжних розмірів обсягів. Це необроблене кількість активацій на кожному шарі CNN, а також їх градієнти. Зазвичай більшість активацій розташовується на найперших шарах згуткової нейромережі. Це слід мати на увазі, так як дані активації потрібні для backpropagation. Однак реалізація, яка запускає CNN тільки під час випробувань, може зменшити кількість активацій шляхом збереження лише поточних з них на будь-якому шарі і відкидання всіх попередніх на шарах нижче;

— з розмірів параметрів. Це числа, які містять параметри мережі, їх градієнти під час зворотного поширення і, як правило, крок кеша. Таким чином, пам'ять для зберігання вектору параметрів повинна бути помножена на коефіцієнт, приблизно рівний 3;

— будь-яка реалізація згуткової нейронної мережі повинна підтримувати різнорідну пам'ять начебто пакета даних зображень, можливо, їх вдосконалені реалізації.

Як тільки отримано приблизний розмір загальної кількості значень (для активацій, градієнтів і іншого), результат необхідно перетворити в Гб. Потрібно взяти кількість значень, помножити його на 4, щоб отримати число байтів, а потім розділити на1024^3 , Щоб отримати пам'ять в гігабайтах. Якщо вийшла мережа не

задовольняє вимогам за розмірами, тоді зменшити її можна шляхом зменшення пакетного розміру, оскільки більшість пам'яті зазвичай споживається активації.

2.8 Підходи до проектування архітектури мережі

Найчастіше при побудові архітектури CNN складають кілька шарів шарів згортки і ректифікації (ReLu), за ними слідують шари ПУЛІНГ; цей шаблон продовжує працювати, доки знімок не буде об'єднано до малого розміру. В деякий момент здійснюється перехід до повнозв'язну верствам. Останній повнозв'язну шар містить вихідну інформацію, наприклад, оцінки класу. Іншими словами, найбільш поширена архітектура CNN відповідає схемі:

$$INPUT \rightarrow [[CONV \rightarrow RELU] * N \rightarrow POOL?]*M \rightarrow [FC \rightarrow RELU]*K \rightarrow FC \quad (2.8)$$

Тут «*» означає повторення, а POOL? вказує на додатковий шар пулінгу. Крім того, $N \geqslant 0$, (зазвичай $N \leq 3$), $M \geq 0$, $K \geq 0$ (зазвичай $K < 3$). Нижче представлені архітектури CNN, організовані за цим зразком:

$$INPUT \rightarrow FC \quad (2.9)$$

реалізує лінійний класифікатор.

Тут $N = M = K = 0$.

$$INPUT \rightarrow CONV \rightarrow RELU \rightarrow FC, \quad (2.10)$$

$$INPUT \rightarrow [\text{CONV} \rightarrow \text{RELU} \rightarrow \text{POOL}] * 2 \rightarrow \text{FC} \rightarrow \text{RELU} \rightarrow \text{FC}. \quad (2.11)$$

Тут один згортковий шар – між кожним шаром пулінгу.

$$INPUT \rightarrow [\text{CONV} \rightarrow \text{RELU} \rightarrow \text{CONV} \rightarrow \text{RELU} \rightarrow \text{POOL}] * 3 \rightarrow$$
$$\rightarrow [\text{FC} \rightarrow \text{RELU}] * 2 \rightarrow \text{FC} \quad (2.12)$$

Тут 2 шари згортки укладаються перед кожним шаром пулінгу.

Це хороша реалізація для великих і більш глибоких мереж, оскільки кілька складених шарів згортки можуть розвивати більш складні властивості вхідних даних перед деструктивною операцією пулінгу.

Як правило, віддають перевагу стеку шарів з невеликими фільтрами, як одному великому рецептивної полю згорткового шару. Припустимо, ви, дотримуючись нелінійність між шарами, викладаєте 3 шари згортки розміру 3 × 3 один на одного. При такому розташуванні, кожен нейрон на першому свёрточном шарі має видом 3 × 3 вхідного обсягу. Нейрони на другому шарі мають видом 3 × 3 першого шару і 5 × 5 вхідного обсягу. Точно так же нейрони на третьому шарі мають видом 3 × 3 другого шару і 7 × 7 вхідного обсягу. Припустимо замість цих трьох згорткових шарів, ми хочемо використовувати тільки один convo шар з рецептивних полями розміру 7 × 7. Нейрони такого шару матимуть рецептивное поле вхідного обсягу, аналогічне просторової протяжності 7 * 7, але з деякими недоліками. По-перше, нейрони будуть обчислювати лінійну функцію вхідного обсягу, в той час як стек з 3 шарів згортки містить нелінійності, що підкреслюють властивості шарів. По-друге, якщо ми припустимо, що весь вхідний обсяг має C каналів, тоді один згортковий шар розміру 7 * 7 буде містити $C * (7 * 7 * C) = 49C^2$ параметрів,

в той час як стек з 3 шарів містить лише $3 * (C * (3 * 3 * C)) = 27C^2$

параметрів. Очевидно, що використання декількох convo шарів з невеликими фільтрами дозволяє підкреслити більш потужні особливості вхідної інформації з меншою кількістю параметрів. Недоліком є більший обсяг використовуваної пам'яті для збереження всіх проміжних результатів.

На практиці використовують те, що краще працює з базою даних ImageNet. Розробникам не слід турбуватися про більш ніж 90% архітектур, адже головне завдання - вибрати ту, яка в даний час найбільш сумісна з ImageNet, скачати попередньо навчену модель і налаштувати її під свої дані. Розробникам практично ніколи не доводиться навчати згорткову нейромережу з нуля.

2.9 Висновки до розділу

В данному розділі було детально розглянуто теорію CNN, розглянуто їх архітектуру, описано принцип роботи шарів згортки, ReLu та пулингу. Крім цього, було розглянуто роботу оптимізатора, який забезпечує оптимізацію параметрів моделі, координуючи прямий і зворотний градієнти мережі для формування оновлень параметрів.

Великим плюсом CNN є те, що вони позбавлені проблем зникання або вибуху градієнтів. Також вони потребують набагато менше параметрів в порівнянні з звичайною нейронною мережею прямого поширення.

3 ОГЛЯД БІБЛІОТЕК, ФРЕЙМВОРКІВ І АРХІТЕКТУР CNN

3.1 Огляд бібліотек і фреймворків, що реалізують алгоритми глибинного навчання

3.1.1 Caffe

Caffe підтримує багато типів машинного навчання, націлених в першу чергу на вирішення завдань класифікації та сегментації зображень. Підтримує згорткові нейронні мережі, RCNN, довгу короткострокову пам'ять і повнозв'язні нейронні мережі. При цьому для прискорення навчання застосовується система графічних процесорів (GPU), що підтримує архітектуру CUDA та застосовує бібліотеку CuDNN від фірми Nvidia.

Caffe дозволяє використовувати готові промислові конфігурації нейронних мереж, що пройшли апробацію. У комплект входить, зокрема AlexNet, яка перемогла в 2012 році в змаганні з розпізнавання зображень ImageNet, і GoogLeNet, яка перемогла в змаганнях ImageNet 2014 року.

Caffe маніпулює блобами - багатовимірними масивами даних, які використовуються в паралельних обчисленнях, які поміщаються в CPU або GPU. Навчання CNN реалізується як паралельні багатопроцесорні обчислення блобів від шару до шару (прямим і зворотним ходом). Solver (вирішувач) координує весь процес навчання - прямий хід від вхідних до вихідних даних, отримання функції помилок, зворотний хід (Метод backpropagation) назад від вихідного шару з використанням градієнтів помилок. При цьому Caffe реалізує різні стратегії навчання для Solvera.

3.1.2 Deeplearning4j

Бібліотека на Java і Scala використовує фреймворк з відкритим вихідним кодом для реалізації розподіленої обробки неструктурованих і слабоструктурованих даних Apache Spark. Є бібліотекою для глибинного навчання загального призначення, призначена для запуску на JVM оточенні. Ядром бібліотеки є блок наукових обчислень написаний на C ++. Дозволяє створювати шари з заданими параметрами. Інтегрований в пакети Hadoop і Kafka. Ліцензії Apache 2.0.

Deeplearning4j – бібліотека програм на мові Java, яка використовується як фреймворк для глибокого навчання. Включає реалізацію обмеженою машини Больцмана, глибокої мережі довіри, глибокого автокодіровщіка, стекового автокодіровщіка з фільтрацією шуму, рекурсивної тензорною нейронної мережі, word2vec, doc2vec, and GloVe. Ці алгоритми включені також у версії бібліотеки, що підтримують розподілені обчислення, інтегровані з архітектурою Apache Hadoop і Spark.

Додаткова бібліотека ND4J відкритого доступу забезпечує обчислення на графічних процесорах з підтримкою CUDA. Крім того, є засоби для роботи з бібліотекою мовою Python через фреймворк Keras.

3.1.3 MatConvNet

MatConvNet - це реалізація з відкритим кодом конволюційних нейронних мереж (CNN) з глибокою інтеграцією в середовище MATLAB. Панель інструментів розроблена з акцентом на простоту та гнучкість. Це розкриває будівельні блоки CNN як прості у використанні функції MATLAB, забезпечуючи підпрограми для

обчислення згортків з банками фільтрів, об'єднання функцій, нормалізацію та набагато більше. MatConvNet можна легко розширити, часто використовуючи лише код MATLAB, що дозволяє швидко прототипувати нові CNN архітектури. У той же час він підтримує ефективність обчислення на процесорі та графічному процесорі, що дозволяють тренувати складні моделі на великих датасетах, таких як ImageNet ILSVRC, що містять мільйони навчальних прикладів.

3.1.4 Neon

Заявляється розробниками як найшвидший фреймворк для CNN і глибинного навчання з підтримкою обчислень на GPU і CPU. Фронтенд виконаний на мові Python, в той час як самі алгоритми реалізовані на спеціально розробленому шейдерний асемблері. Розроблено компанією Nervana Systems, яка була куплена Intel.

3.1.5 TensorFlow

TensorFlow – відкрита програмна бібліотека для машинного навчання, розроблена компанією Google для вирішення завдань побудови і тренування нейронної мережі з метою автоматичного знаходження та класифікації образів, досягаючи якості людського сприйняття. Застосовується як для досліджень, так і для розробки власних продуктів Google. Основний API для роботи з бібліотекою реалізований для Python, також існують реалізації для C Sharp, C ++, Haskell, Java, Go і Swift. Підтримує обчислення на CPU, GPU і спеціально розробленими компанією Google TPU (Tensor processing units).

3.1.6 Theano

Theano - бібліотека чисельного обчислення в Python. Обчислення в Theano виражаються NumPy-подібним синтаксисом і компілюються для ефективних паралельних обчислень як на звичайних CPU, так і на GPU.

Theano є проектом з відкритим вихідним кодом, основним розробником якого є група машинного навчання в Монреальському університеті.

Дозволяє користувачеві писати символічні формальні математичні вирази, з яких автоматично генерується похідний код. Таким чином користувачеві не потрібно програмувати градієнти або зворотне поширення помилки. Такі вираження автоматичний компілюються в шейдерний код для CUDA для оптимізації обчислень на GPU.

28 вересня 2017 року було оголошено про припинення роботи над проектом після виходу релізу 1.0, при цьому обіцяно збереження його мінімальної підтримки протягом одного року

3.1.7 Torch torch.ch

Фреймворк для наукових обчислень з широкою підтримкою машинного навчання, написаний на C і Lua. Фреймфорк, на даний момент використовується дослідним підрозділом в області штучного інтелекту компанії Facebook, а також компанією Twitter в реалізаціях систем автоматичної класифікації користувацького контенту.

Видно, що основний підхід полягає в написанні високоефективного ядра бібліотеки виконуючого сам алгоритм на мові низького рівня (C, C ++, асемблер). Потім для забезпечення зручності використання і читання коду користувачеві надається фронтенд на мові високого рівня, звідки можна, користуючись більш зручним синтаксисом, робити виклики в ядро бібліотеки. Підтримується розпаралелювання обчислень засобами CUDA і OpenMP.

3.2 Критерії оцінки технологій

З урахуванням наведених вище обмежень і вимог, можна сформулювати наступні критерії оцінки використовуваних технологій (табл. 3.1):

– ядро має ефективно використовувати всі можливості сучасних графічних процесорів для розпаралелювання обчислень на навчальному сервері.

– технологія повинна бути добре відомою, підтримуваної, документованої мати розвинуте активна спільнота розробників. Це необхідно для простоти отримання технічних рекомендацій по реалізації демонстраційного проекту.

– технологія повинна мати можливість розгортання на архітектурі ARM для того, щоб була можливість розгортання мережі на мобільних пристроях, відеореєстраторах і т.д.

Таблиця 3.1 - Відповідність популярних бібліотек критеріям оцінки

Бібліотека (фреймворк)	Оптимізація під GPU	Популярність	Простота розгортання проекту на ARM архітектурі
Caffe	+	+	+
Deeplearning4j	+	+	-
MatConvNet	+	+ -	-
Neon	+	+	-
TensorFlow	+	+	-
Theano	+	+	+
Torch	+	+	-

Таким чином, проаналізувавши велику кількість статтей, було прийнято рішення використовувати бібліотеку TensorFlov. Для виконання проекту потрібно було виконати фронтенд бібліотеки на мові високого рівня і реалізацію механізму викликів в код ядра бібліотеки.

3.3 Огляд деяких існуючих архітектур CNN

Розглянемо найбільш популярні архітектури згуткових нейронних мереж:

- LeNet;
- AlexNet;
- ZF Net;

– GoogLeNet;
– VGGNet;
– ResNet.

3.3.1 LeNet

Перше успішне застосування CNN вдалося розробити Яну Лекуну в 1990-і роки. Архітектура LeNet (рисунок 3.1) використовувалася для сканування поштових індексів, цифр і т. п.

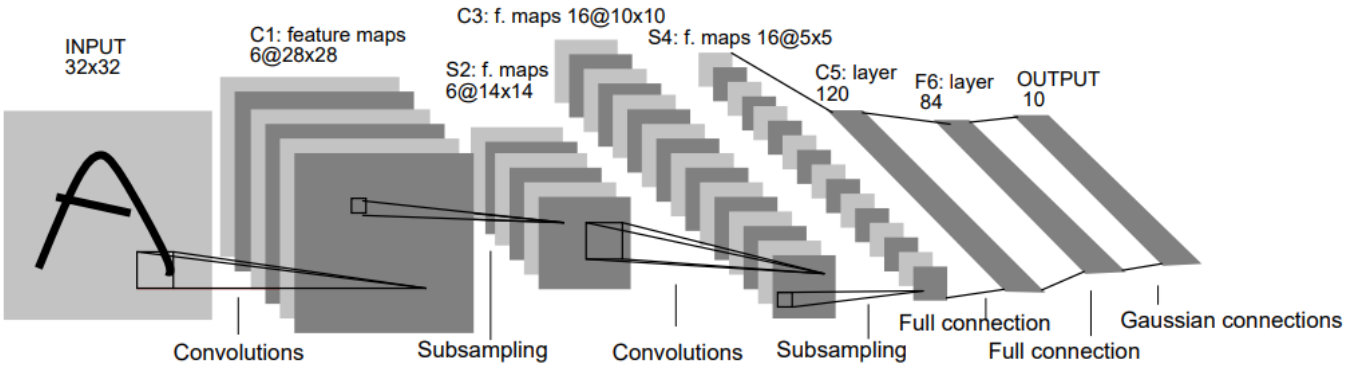

Рисунок 3.1 – Архутектура нейронної мережі LeNet-5

3.3.2 AlexNet

Перша робота, яка популяризувала згорткові нейронні мережі в напрямку комп'ютерного зору, розроблена Alex Krizhevsky, Ilya Sutskever і Geoff Hinton. AlexNet була протестована на змаганні ImageNet ILSVRC в 2012 році і значно випередила конкурента на другому місці (відсоток помилок: 16% проти 26%). Архітектура мережі була дуже схожа на архітектуру LeNet, але була глибше, більше

і використовувала послідовності згорткових шарів (до цього було стандартною практикою використовуватися тільки комбінацію, в якій вісля згорткового шару відразу слідував шар підвибірки). Вона зображена на рис. 3.2.

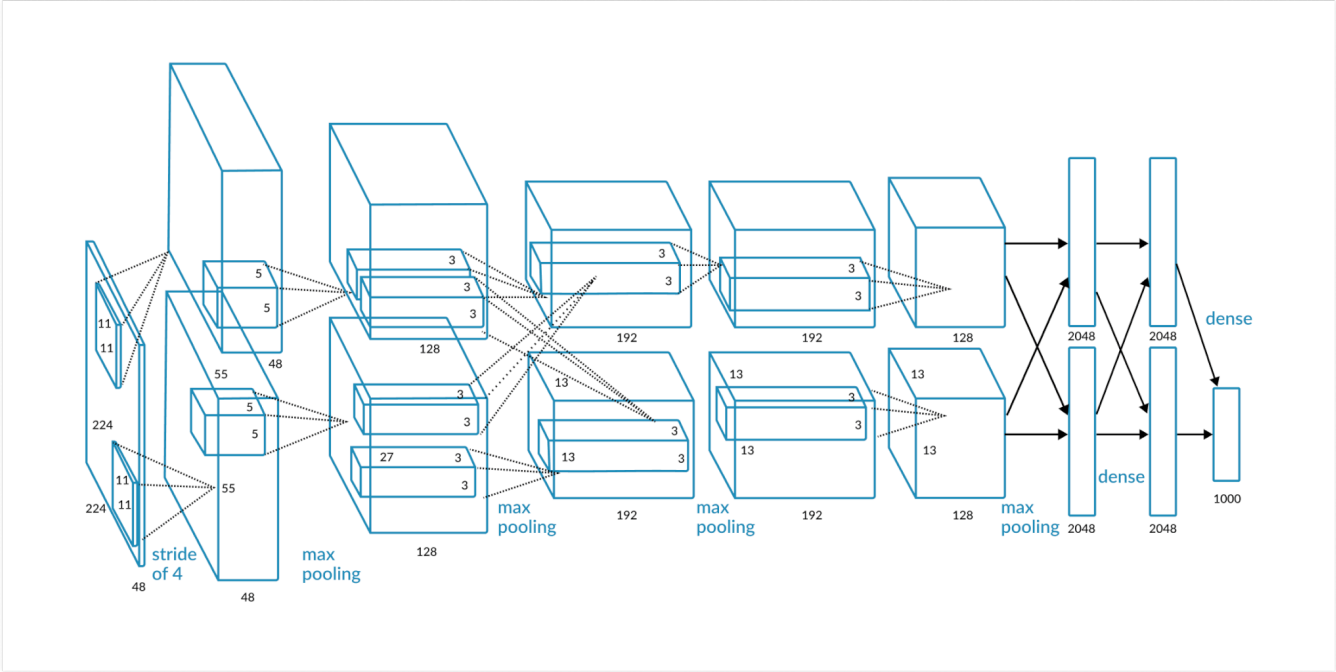

Рисунок 3.2 – Архітектура нейронної мережі AlexNex

3.3.3 ZF Net

А ось переможцем ILSVRC 2013 стала CNN ZF Net (абревіатура від Zeiler і Fergus), розроблена Метью Зеллером і Робом Фергюсом. Дана архітектура була поліпшеною версією AlexNet: тут збільшили розміри середніх згуткових шарів і зменшили крок і розмір фільтра на першому шарі, що показано на рисунку 3.3.

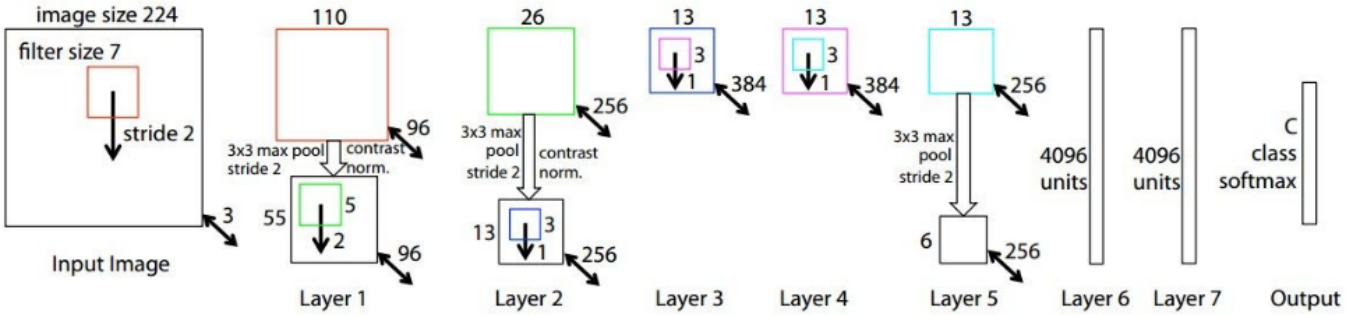

Рисунок 3.3 – Архітектура нейронної мережі ZF Net

3.3.4 GoogLeNet

GoogLeNet збільшив очікувану середню точність виявлення об'єктів до 0.439329, і знизив похибку класифікації до 0.06656, найкращого результату на той момент. Його мережа застосовувала понад 30 шарів. Число параметрів склало всього 4млн. Ця продуктивність згорткових нейронних мереж у завданнях ImageNet була близькою до людської. Архітектура данної нейронної мережі наведена на рисунку 3.4.

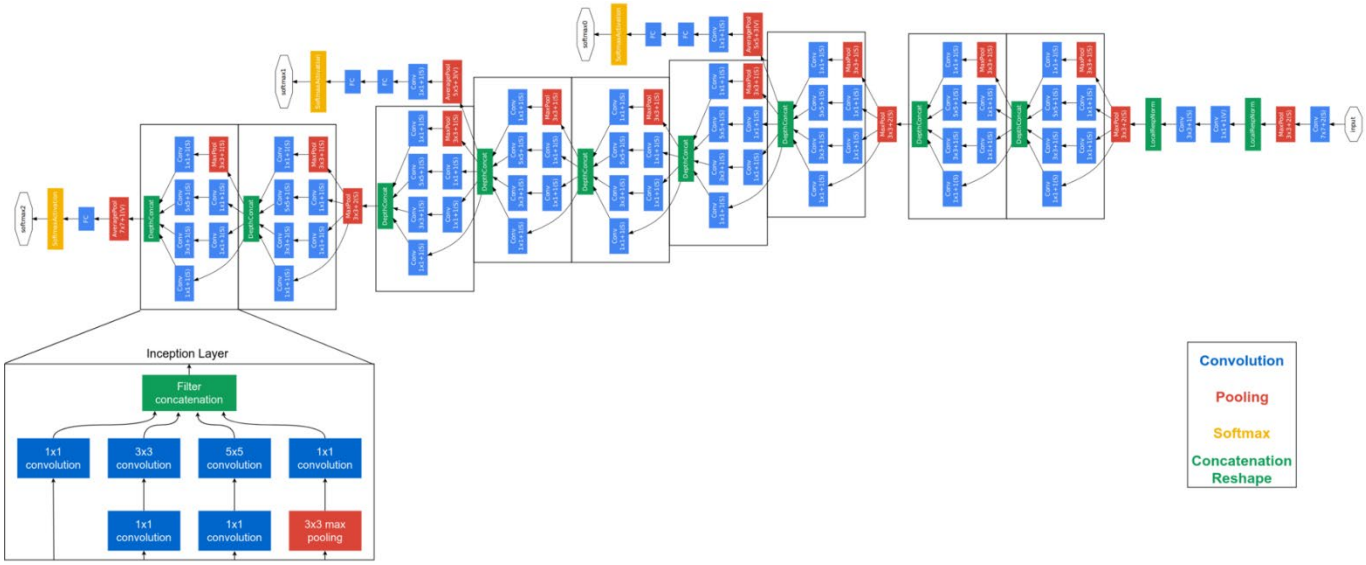

Рисунок 3.4 – Архітектура GoogLeNet

3.3.5 VGGNet

На ILSVRC 2014 розташувалася мережа Карена Симоняна і Ендрю Ціссермана. Було продемонстровано, що глибина є дуже важливою для швидкодії. CNN складається з 16 згорткових і повнозв'язних шарів Використовується згортка 3 × 3 і пулінг 2 × 2 від початку до кінця. Архітектура VGGNet показана на рисунку 3.5.

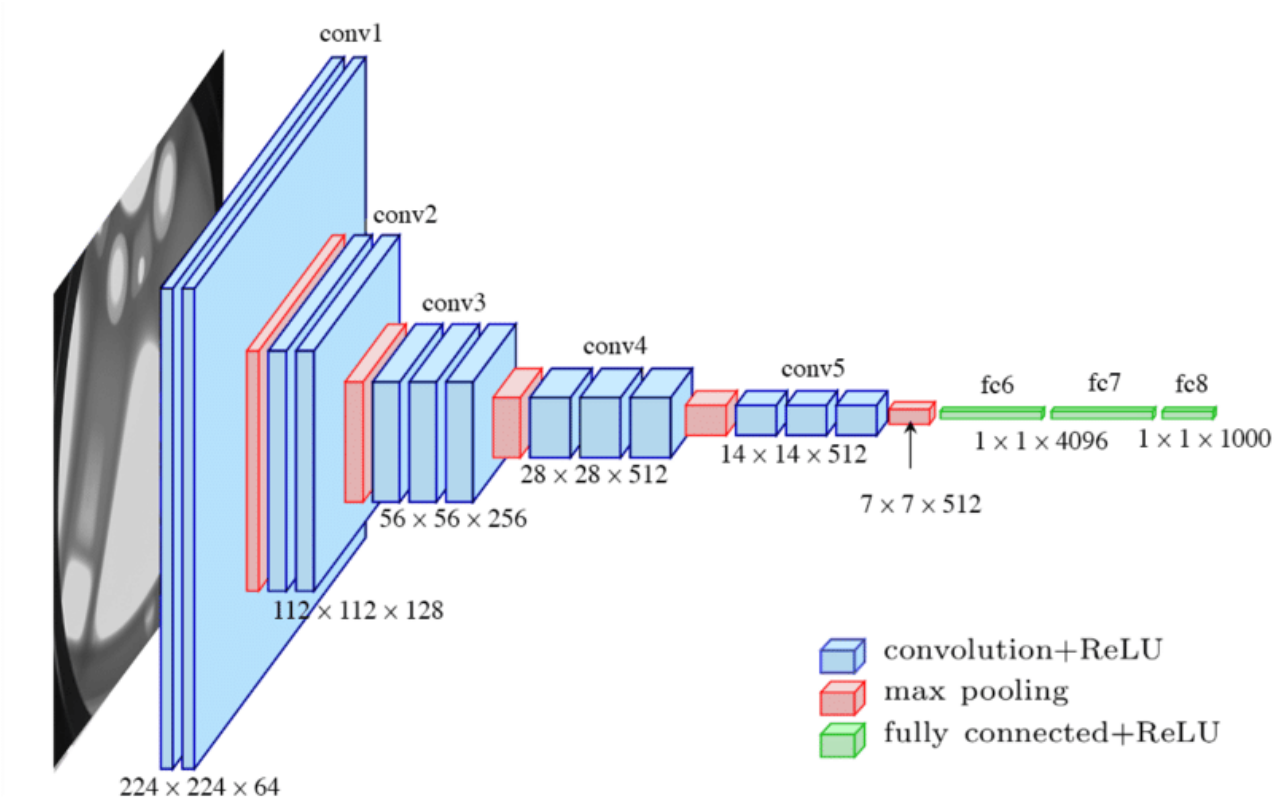

Рисунок 3.5 – Архітектура VGGNet

3.3.6 ResNet

Residual Network розроблена Каймінгом Хе і іншими, стала переможцем ILSVRC. Інтенсивно використовує пакетну нормалізацію і спеціальні skip-connections. В кінці немає повнозв'язних шарів. ResNet на сьогодні є справжнім витвором мистецтва в світі CNN і використовується найбільш часто.

ЇЇ архітектрура показана на рисунку 3.6.

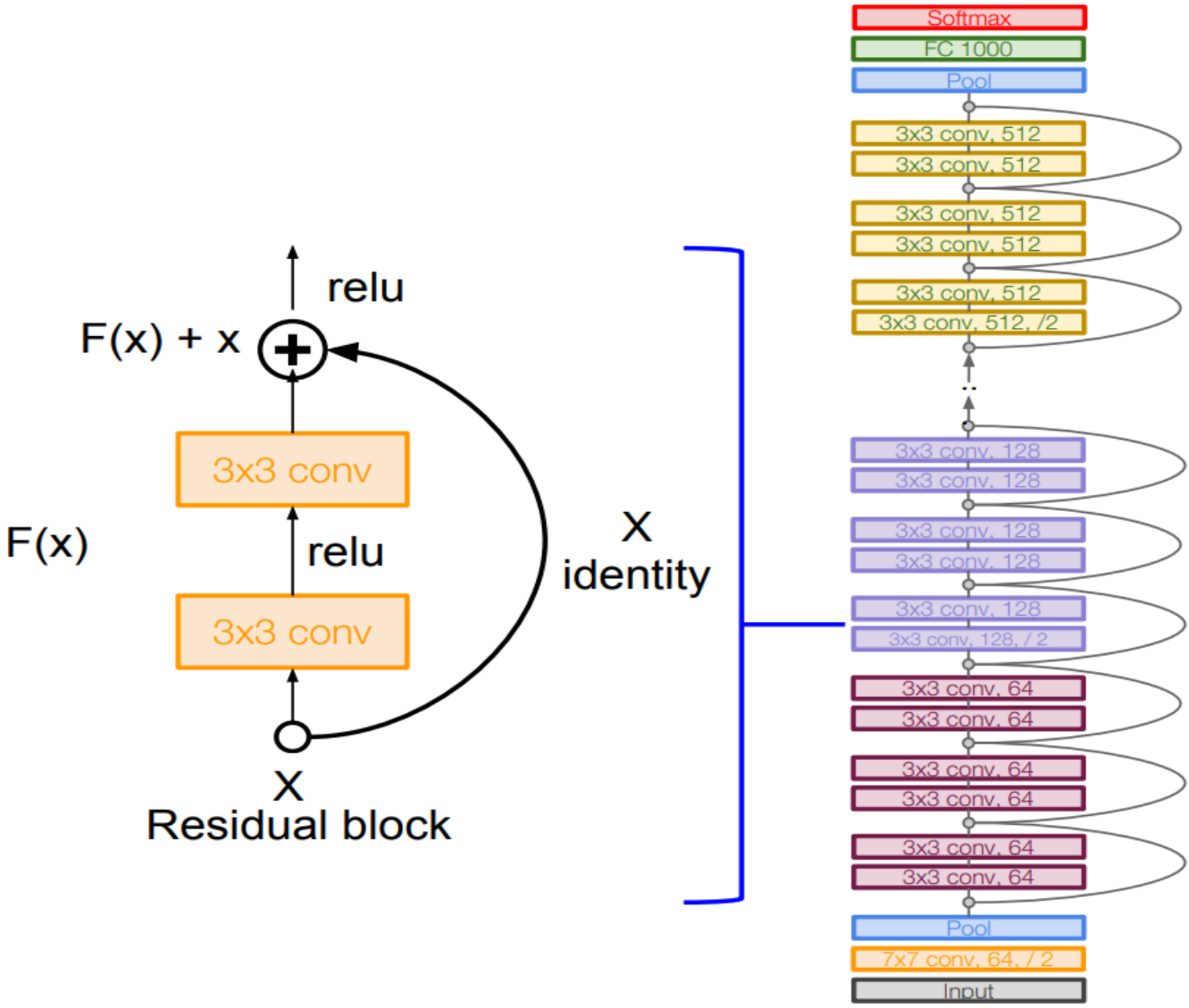

Рисунок 3.6 – Архітектура ResNet

3.4 Висновки до розділу

У цьому розділі подано опис популярних фреймворків, розглянуто їх плюси та мінуси. Після аналізу статтей та готових рішень було вирішено обрати фреймворк TenzorFlow, адже у нього добре описана документація.

При виборі типу гроткової мережі вибір впав на ResNet, адже вона на сьогодні є справжнім витвором мистецтва в світі CNN і використовується найбільш часто.

4 РОЗРОБКА СИСТЕМИ

4.1 Підготовка даних до мережі

Данні для нейронної мережі було взято з автосимулятора “City Car Driving Simulator”. На рисунку 4.1 представлено приклад таких данних

Рисунок 4.1 – Приклад вхідних данних

4.2 Архітектура програми

За основу данної роботи було взято проект MultiNet, в якому було покращено роботу присутніх модулів, а токож створено новий модуль для розпізнавання дорожніх знаків. Для реалізації було вибрано нейронні мережі ResNet. Програма складається з декількох CNN, кожна з яких відповідає за певну частину роботи. Так,

на вході ми маємо CNN для передобробки зображення, тобто приведення зображення в потрібний формат, а також локалізації областей з потрібними для розпізнавання об’єктами. Далі ця нейронна мережа передає данні іншим підмережам для розпізнавання потрібних образів. Одна з цих підмереж відповідає за розпізнавання дорожних знаків, друга – за дорожню розмітку, а третя – за розпізнавання автомобілів. Архітектура програми наведена на рисунку 4.2.

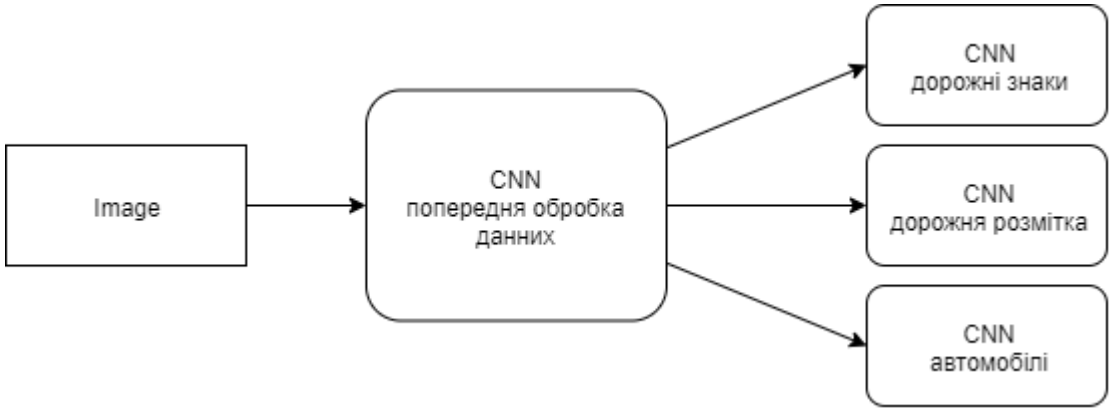

Рисунок 4.2 – Архітектура програми

4.3 Алгоритм навчання мережі

Для автоматизації навчання розроблено алгоритм, який генерує послідовність зображень з відеофайлу, кожен з яких відповідає одному кадру відеопотоку гри з роздільною здатністю 1366x768 pixels.

Щоб прискорити роботу і полегшити пошук його параметрів, зображення приводиться до розмірів 1248 x 384 пікселів.

Таким чином, набір зображень, створених з 15-секундних відео-файлів з 60 FPS використовується для навчання мережі, яка дає приблизно 900 кадрів для підготовки мережі. Експериментально встановлено, що функція втрати надійно переходить до ассипту після 6 епох навчання, за умови, що початковий коефіцієнт швидкості

знаходиться в діапазоні від 10^{-6} до 10^{-5} і змінюється відповідно до методу стохастичного градієнтного спуску.

Крім прикладів із зображенням об'єктів, який мережа повинна навчитися розпізнавати (positive samples) потрібно показувати їй зображення, які не містять ці об'єкти (negative samples).

4.4 Знаходження архітектури мережі

Знаходження підходящої нейромережі архітектура та хитрості, незважаючи на розроблений математичний апарат, сьогодні не так багато , як наступні послідовність інструкцій або наступних загальних рекомендацій. Специфіка роботи з нейромережами полягає в тому, що результат конкретної нейронної мережі не легко передбачити і навіть невеликі зміни в налаштуваннях, не пов'язаних з архітектурою мережі (наприклад , розмір ядра розшарування, кількість ознак, створених або функція активації) може несподівано впливати на мережу.

У зв'язку з цим вирішення конкретної проблеми є ітеративний процес, що складається з наступних кроків:

1. Проектування мережної архітектури.
2. Реалізація мережі.
3. Настроювання мережі.
4. Навчання.
5. Тестування.

У випадку, якщо результати тесту є незадовільними повністю (відбувається перепідготовка мережі, або мережа не може знайти локальний максимум продовжує

змінювати вагу в діапазоні значень і т. д. D.), потім нова ітерація відновиться з кроком 1.

Якщо мережа в цілому реагує на дані, але точність і похибка не досягають цільових параметрів, можна припустити, що архітектура підхоплюється, а нова ітерація оновлюється з кроку 3.

Незважаючи на це, процес проектування мережі не хаотичний і не випадковий. У цій галузі знань дуже багато практичних даних. Існує набір рекомендацій, спрямованих на прискорення пошуку оптимальної архітектури та налаштувань.

Це також сприяє наявності великої кількості відомих архітектур , які успішно виконують завдання класифікації зображень і навіть локалізуючи їх на шарі даних.

4.5 Навчання нейронної мережі

Для навчання нейронної мережі було використано декілька разних датасетів. Так, для навчання нейронної мережі, що відповідає за розпізнавання дорожніх знаків, був взятий датасет GTSRB – датасет для розпізнавання дорожніх знаків в Німеччині. Данний датасет складається більш ніж з 50 тисяч зображень різних дорожніх знаків. Процес навчання данного модуля тривав 27 днів 3:47:06. Модуль по розпізнаванню дорожніх знаків розпізнає дорожній знак, обрамлює його, а також виводить назву данного знаку на екран. Результати тестування данного модуля можна побачити на рисунку 4.3. Результат роботи модуля по розпізнаванню дорожних знаків показано на рисунку 4.4.

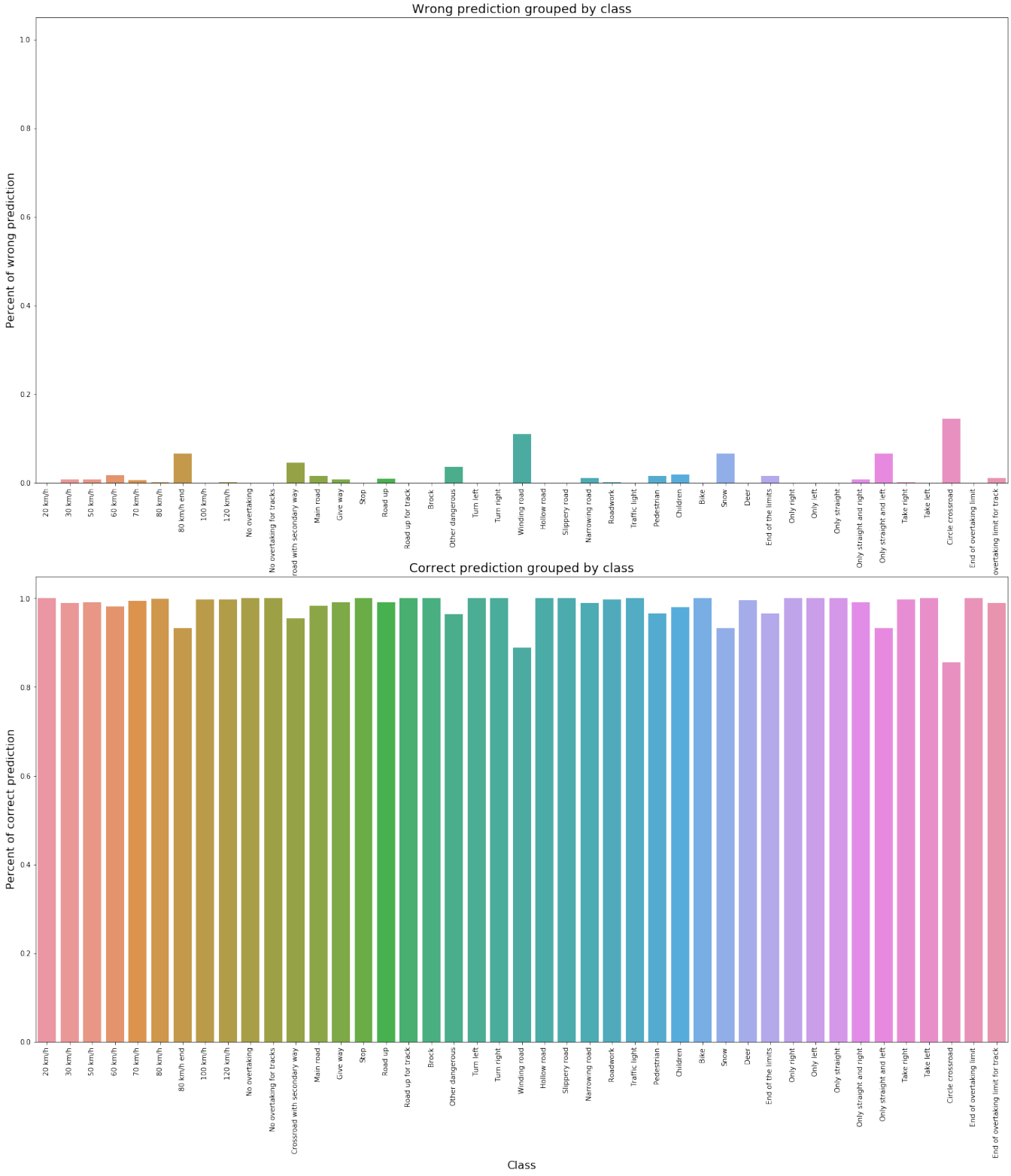

Рисунок 4.3 – Результати тестування модуля розпізнавання знаків

Рисунок 4.4 – Результат роботи модуля розпізнавання дорожних знаків

Для навчання модуля по розпізнаванню дороги було взято датасет з вибіркою різних доріг при різних погодніх умовах та різному освітленню. Він складається з порядка 37 тисяч різних зображень. На процес навчання було витрачено 22 дні 00:03:12 годин. Данний модуль розпізнає та візуалізує дорогу на зображенні. Приклад такого виводу можна побачити на рисунку 4.5.

Рисунок 4.5 – Приклад виводу модуля розпізнавання дороги після процесу навчання

Для навчання модуля розпізнавання автомобілей було обрано датасет з різними класами ТЗ. Данний датасет має в собі більше 60 тисяч різних ТЗ, при чому він складається з підвибірок різних типів ТЗ, таких як легкові автомобілі, буси, вантажні автомобілі і т.д. Процес навчання зайняв 31 день 10:14:34 годин. Данний модуль

знаходить на зображенні автомобіль та обрамлює його. Результат роботи данного модулю після тренування можна побачити на рисунку 4.6.

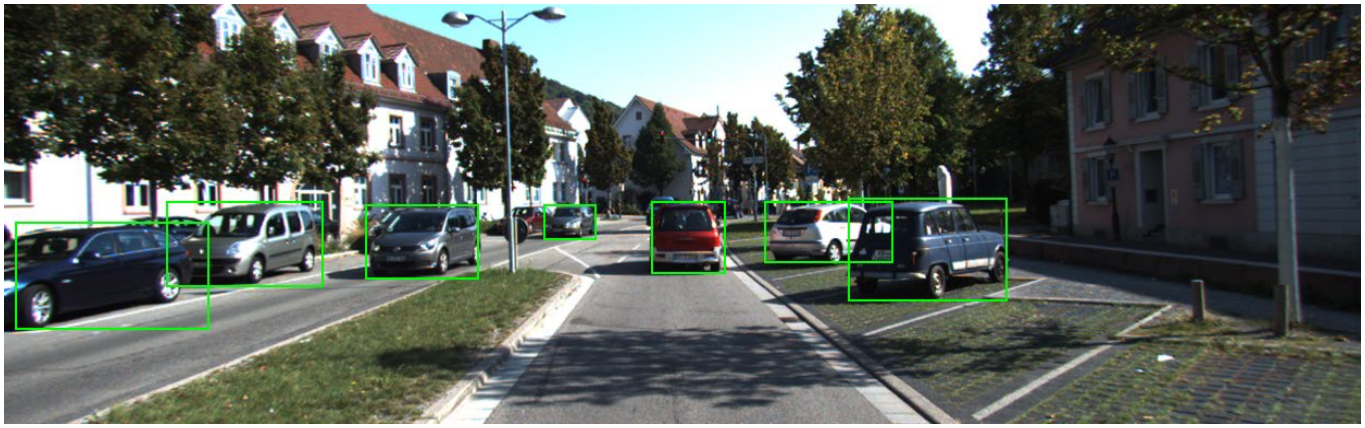

Рисунок 4.6 – Робота модуля по розпізнаванню ТЗ

4.6 Отримані результати

Результатом роботи програми є видозмінене зображення з виділеними на ньому розпізнаними образами. Приклад такого виводу можна побачити на рисунку 4.7.

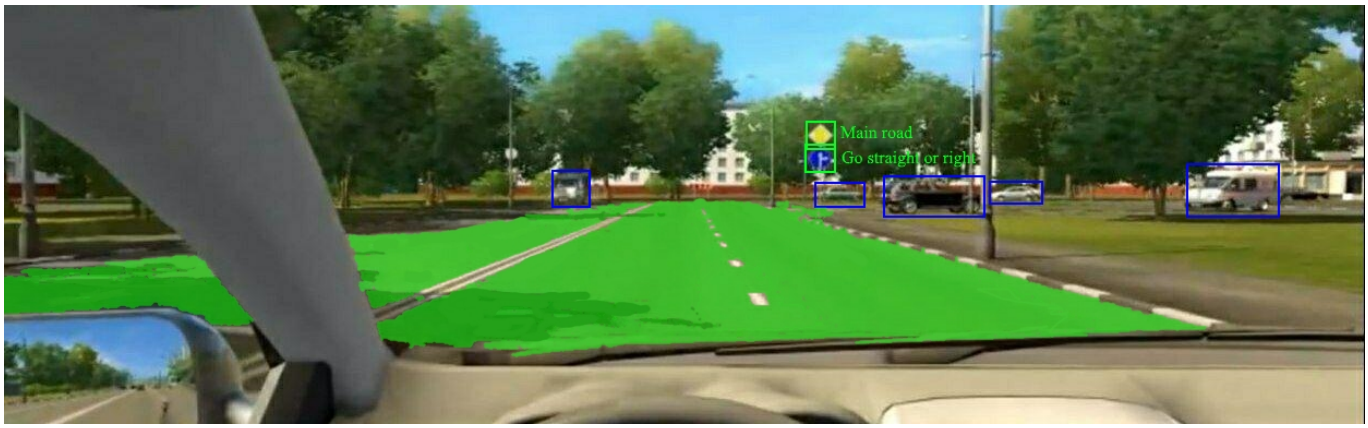

Рисунок 4.7 – Результат роботи програми

Програма може працювати з зображеннями, з відеофайлами та з потоковим зображенням з автосимулятора. Конфігурація для тестування програми була така: процесор Intel Core i5 6600k, відеокарта Nvidia GTX 1070, об’єм оперативної пам’яті 16 Gb DDR4. З данною конфігурацією було отримано продуктивність на рівні 26 кадрів за сакунду, що є досить задовільним результатом.

4.7 Висновки до розділу

В данному розділі було описано програмну реалізацію системи розпізнавання образів на дорозі. Данна програма складається з трьох згорткових мереж ResNet101. Перша нейронна мережа – модуль по розпізнаванню дорожних знаків, друга – модуль по розпізнаванню ТЗ, а третя – модуль по розпізнаванню дороги. Швидкодія склала приблизно 26 кадрів в секунду Для забезпечення більшої швидкодії можна використовувати нейтонну мережу ResNet50 або інші нейронні мережі, наприклад VGG16, але при цьому буде погіршуватись точність розпізнавання.

5 РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

У даному розділі буде розглянуто ключові особливості розробленої системи як майбутнього стартап-проекту. Проект розглядатиметься як система автономного управління автомобілем.

5.1 Опис ідеї проєкту

Спочатку проаналізуємо та подамо у вигляді таблиці зміст ідеї стартап-проекту, можливі напрямки застосування та основні вигоди, які може отримати користувач товару. Ці характеристики стартап-проекту зображено в таблиці 5.1.

Таблиця 5.1 - Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Програмний забезпечення для автономного керування автомобілем.	1. Застосування як допоміжної системи в парі з людиною.	Можливість у разі потреби взяти керування автомобілем програмним забезпеченням, а також інформування людини про порожню обстановку та знаки.
	2. Застосування як системи автономного керування автомобілем без участі людини.	Можливість повного автономного керування автомобілем.

Тепер зробимо аналіз потенційних техніко-економічних переваг ідеї порівняно із пропозиціями конкурентів. Результати аналізу зображено в таблиці 5.2.

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	Товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конкурент 1	Конкурент 2			
1.	Ціна	2000$/рік	4000$/рік	5000$/рік			+
2.	Прибутки	30000$/рік	40000$/рік	20000$/рік		+	
3.	Контроль якості	Аналітики та програмісти	Аналітики, програмісти та деякі клієнти	Програмісти			+
4.	Динаміка галузі	Швидка	Повільна	Швидка		+	
5.	Постійні витрати	10000$/рік	20000$/рік	15000$/рік			+
6.	Змінні витрати	5000$ - 10000$/рік	1000$ - 2000$/рік	2000$ - 5000$/рік	+		
7.	Патенти на продукти	Немає	Патент на кожний проект	Декілька патентів на винахід	+		

Продовження таблиці 5.2

№ п/п	Техніко-економічні характеристики ідеї	Товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конку-рент 1	Конку-рент 2			
8.	Гнучкі ціни	Ціна єдина	Ціна варію-ється з року в рік	Ціна єдина		+	
9.	Законо-давчі обмеження	Обмеження на використання приватних даних (голоса) людини – GDRP	Немає	Обме-ження на кіль-кість розро-бників			+

5.2 Технологічний аудит ідеї проєкту

Визначимо технологічну здійсненність ідеї проекту за допомогою аналізу таких складових, як технології, за якою буде виготовлено товар згідно ідеї проекту, існування таких технологій, чи їх необхідно розробити / доробити, доступність таких технологій авторам проекту. Результати даного аналізу зображено в таблиці 5.3.

Таблиця 5.3 – Технологічна здійсненність ідеї проекту

Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
Програмне забезпечення для автономного керування автомобілем.	Технологія проектування та розробки системи для розпізнавання образів зображень на дорозі за допомогою OpenCV.	Так	Дані технології доступні, але потрібно використовувати багато залежностей.
	Технологія проектування та розробки класифікаторів sklearn.	Так	Дані технології доступні, однак, штатного функціоналу все ще не вистачає для вирішення поставлених задач.
	Технологія здійснення порівняльного аналізу sklearn. .neighbors.LocalOutlierFactor	Так	Дані технології доступні.
Обрана технологія реалізації ідеї проекту: технологія вилучення ознак з зображення, класифікації та порівняння (sklearn).			

5.3 Аналіз ринкових можливостей запуску стартап-проєкту

Проведемо аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку. Результати даного аналізу зображено в таблиці 5.4.

Таблиця 5.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	*Показники стану ринку (найменування)*	*Характеристика*
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн/ум.од	300 000
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Висока точність розпізнавання, швидкодія, невибагливість до ресурсів
5	Специфічні вимоги до стандартизації та сертифікації	Немає
6	Середня норма рентабельності в галузі (або по ринку), %	80

Таким чином, за попереднім оцінюванням, ринок є привабливим для входження.

Надалі визначимо потенційні групи клієнтів, їх характеристики, та сформуємо орієнтовний перелік вимог до товару для кожної групи. Ці дані зображено в таблиці 5.5.

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*
1	Допоміжна система керування автомобілем	Фізичні особи, малий, середній та великий бізнес	Цільовій аудиторії потрібен інструмент допомоги при керуванні ТЗ.	Клієнти прагнуть забезпечити безпечні умови при їзді на дорозі.
2	Система автономного керування автомобілем	Фізичні особи, малий, середній та великий бізнес	Цільовій аудиторії потрібен інструмент, який у будь-який момент може повністю взяти керування ТЗ на себе у разі потреби або по бажанню людини	Клієнти прагнуть мати можливість передати керування автомобілем бортовому комп’ютеру за потреби або по бажанню.

Після визначення потенційних груп клієнтів проведемо аналіз ринкового середовища: складемо таблиці факторів, що сприяють ринковому впровадженню проекту (таблиця 5.6), та факторів, що йому перешкоджають (таблиця 5.7).

Таблиця 5.6 – Фактори загроз

№ п/п	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1	Відсутність попиту	Цільова аудиторія може не оцінити переваги продукту, або ж у цілому відмовитися від ведення нарад	Акцентувати увагу на клієнтах, що вже скористалися продуктом, якщо такі є, навести інфографіку результативності (очікувану), запропонувати знижку потенційному клієнту в рамках тендеру.
2	Неточне розпізнавання	Особливості освітлення та дорожної обстановки можуть призвести до зниження точності розпізнавання знаків та інших учасників дорожнього руху	Розробка і випуск оновлення ядра системи, де виправлена ця проблема.

Таблиця 5.7 – Фактори можливостей

№ п/п	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1	Кобрендінг	Пропозиція від певної компанії, що спеціалізується на системах автономного керування ТЗ, розробити спільний продукт	Виділення частини штату на реалізацію проекту, підготовка акційних пропозицій по переходу на новий продукт існуючим клієнтам.

Надалі проведемо аналіз пропозиції: визначимо загальні риси конкуренції на ринку. Результати даного аналізу зображені в таблиці 5.8.

Таблиця 5.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. Чиста конкуренція	Гравці ринку не мають явних переваг один над одним	Більш вигідні умови на тендерах, агресивний маркетинг
2. Регіональна конкуренція	Гравці ринку – інтернаціональні підприємства	Вихід на ті ринки, які ще не зайняті конкурентами
3. Внутрішньогалузева конкуренція	Гравці ринку знаходяться в одній галузі – розробці ПЗ	
4. Товарно-видова конкуренція	Усі продукти гравців ринку мають одне призначення	Розробка найбільш інтуїтивного інтерфейсу, розробка унікальних мовленнєвих пакетів, оптимізація алгоритмів (щоб аналіз проходив швидше, ніж у конкурентів)
5. Конкурентні переваги неціновi	Продукти відрізняються гнучкістю, функціоналом (незначно) і надійністю.	У маркетингу неявно порівнювати власний продукт з іншими, робити вигідні цінові пропозиції
6. Марочна конкуренція	Значна увага приділяється бренду, що розробив продукт	Кобрендінг

Тепер визначимо та обґрунтуємо фактори конкурентоспроможності, які зображені в таблиці 5.9.

Таблиця 5.9 – Обґрунтування факторів конкурентоспроможності

№ п/п	*Фактор конкурентоспроможності*	*Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)*
1	Невибагливість до апаратних ресурсів (серверів). А отже дешевизна апаратних ресурсів, потрібних для нашої системи	В продукті використане поєднання класичних методів голосового розпізнавання та методів Машинного навчання
2	Швидкодія	В продукті використане поєднання класичних методів голосового розпізнавання та методів Машинного навчання
3	Інтеграція	Продукт може бути використаний в будь-якому веб-сайті захищеному протоколом ssl. Продукт не потребує придбання спеціалізованого апаратного забезпечення
4	Модульність	Замовник може обирати тип продукту (веб-додаток, або мобільний-додаток)
5	Гнучкість	Кожен замовник має можливість замовити розширення функціоналу продукту під його конкретні задачі
6	Голосова аутентифікація	Клієнти замовника зможуть швидше увійти до системи

Таблиця 5.10 – Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складові аналізу	Динаміка галузі, продуктова лінія, бар'єри проникнення	Наявність товарних знаків, доступ до ресурсів, патенти на продукти	Концентрація постачальників, диференціація витрат	Рівень чутливості до зміни цін, прибутки, контроль якості	Ціна, лояльність споживачів
Висновки:	Конкуренція не є інтенсивною, адже даний ринок ще ніким не зайнятий.	Для входу на ринок необхідно створити товарний знак та написати бета-версію програмного продукту. На даний момент потенційних конкурентів немає.	Постачальники не диктують умови роботи на ринку, бо програмному продукту не потрібно постачання.	Клієнти диктують умови роботи на ринку, бо вони є єдиним джерелом прибутку компанії.	При наявності товарів замінників необхідно буде зменшувати ціну програмного продукту чи створювати ПЗ для інших технічних систем.

За визначеними факторами конкурентоспроможності проведемо аналіз сильних та слабких сторін стартап-проекту. Результати даного аналізу зображено в таблиці 5.11.

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін системи «BioM»

№ п/п	*Фактор конкурентоспроможності*	*Бали 1-20*	*Рейтинг товарів-конкурентів у порівнянні з BioM*						
			–3	*–2*	*–1*	*0*	*+1*	*+2*	*+3*
1	Інтеграція	15					*		
2	Модульність	16			*				
3	Гнучкість	18			*				

Тепер проведемо SWOT-аналіз на основі виділених загроз і можливостей, та сильних і слабких сторін проекту. SWOT-матриця зображено в таблиці 5.12.

Таблиця 5.12 – SWOT-аналіз стартап-проекту

Сильні сторони: автономне керування ТЗ, швидкодія	Слабкі сторони: Інтеграція має пройти із залученням розробників на стороні замовника
Можливості: Кобрендінг	Загрози: Неточність розпізнавання, відсутність попиту

На основі SWOT-аналізу розробимо альтернативи ринкової поведінки для виведення стартап-проекту на ринок та орієнтований оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок. Дані альтернативи зображено в таблиці 5.13.

Таблиця 5.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	*Альтернатива (орієнтовний комплекс заходів) ринкової поведінки*	*Ймовірність отримання ресурсів*	*Строки реалізації*
1	Реалізація можливості використання системи не тільки для допомоги в керуванні, а й автономного керування ТЗ	Середня	18 місяців

2	Створення системи емоційного розпізнавання людини та її стану здоров'я	Висока	24 місяці
3	Розробка MVP	Висока	12 місяців

Серед даних альтернатив було обрано третю альтернативу, адже строки її реалізації найменші та є ймовірність отримання ресурсів.

5.4 Розроблення ринкової стратегії проєкту

Для розроблення ринкової стратегії першим кроком необхідно описати цільові груп потенційних споживачів, які можна побачити в таблиці 5.14.

Таблиця 5.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1.	Фізичні особи	Середня	5-10 тис в рік	Середня	Середня
2.	Малий бізнес	Середня	5-10 підприємств в рік	Середня	Середня
2.	Середній бізнес	Готові	5-10 підприємств в рік	Слабка	Середня
3.	Великий бізнес	Готові	3-5 закладів в рік	Слабка	Складна
Було обрано цільову групу підприємств групи малого бізнеса.					

Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку, яку ображено в таблиці 5.15.

Таблиця 5.15 – Визначення базової стратегії розвитку

Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Концентрація на потребах одного цільового сегменту – системі допомоги водію.	Створений продукт є дешевим у використанні та іноваційним	Стратегія спеціалізації.

Наступним кроком є вибір стратегії конкурентної поведінки, яку зображено в таблиці 5.16.

Таблиця 5.16 – Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохід-цем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
Так.	Компанія буде шукати нових споживачів, але і, за потреби, буде намагатися забирати існуючих у конкурентів.	Компанія, за потреби, буде копіювати характеристики конкурентів.	Стратегія заняття конкурентної ніші.

Тепер розробимо стратегію позиціонування, що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект. Її зображено в таблиці 5.17.

Таблиця 5.17 – Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспро-можні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
Розпізнавання дорожньої обстановки має бути точним. Система є щвидкою	Проведен-ня крупних оновлень (оптимі-зація розрахун-ків), створення додатков-ого функціона-лу.	Товар є іноваційним (в тренді) та дешевим у використанні порівняно з альтернативами	Швидкий, невибагливий до ресурсів, допомагає в керуванні ТЗ, програма працює в режимі онлайн.

5.5 Розроблення маркетингової програми стартап-проєкту

Сформуємо маркетингову концепцію товару, який отримає споживач. В таблиці 5.18 зображено результати попереднього аналізу конкурентоспроможності товару.

Таблиця 5.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1.	Точне розпізнавання дорожньої обстановки.	Точне розпізнавання дорожньої обстановки на основі знаків, розмітки та інших учасників ДР.	Доступність для компаній з невеликим капіталом.
3.	Швидкодія	Швидкодія системи	Більша швидкість

Надалі розробимо трирівневу маркетингову модель товару: уточнимо ідею продукту, його фізичні складові, особливості процесу його надання. Дана модель зображена в таблиці 5.19.

Таблиця 5.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
I. Товар за задумом	Програмний продукт – система автономного керування ТЗ, яка дозволяє користувачу частково або повністю передати керування автомобілем автопілоту.
II. Товар у реальному виконанні	Властивості / характеристики: 1. Можливість роботи системи в режимі допомоги (виведення інформації з дорожніх знаків, екстрене гальмування, контроль полоси) 2. Можливість повність автономної їзди
	Якість: програмний продукт пройшов всі етапі тестування та готовий до використання.
	Файл з розширенням “.ру”, віртуальне середовище.

Продовження таблиці 5.19

Рівні товару	Сутність та складові
III. Товар із підкріпленням	Спеціаліст із впровадження встановлює ПЗ.
	Відділ розробки підтримує життєдіяльність ПЗ.
Захист програмного продукту буде організовано за допомогою ноу-хау.	

Тепер визначимо цінові межі, якими необхідно керуватись при встановленні ціни на потенційний товар, яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів. Аналіз проводився експертним методом і його результати зображено в таблиці 5.20.

Таблиця 5.20 – Визначення меж встановлення цін

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
3000-5000 $/рік	5000-6000 $/рік	12000-50000 $/рік	Нижня межа – 3000 $/рік, верхня межа - 5000 $/рік

Надалі визначимо оптимальну систему збуту, в межах якого приймається рішення. Дану систему зображено в таблиці 5.21.

Таблиця 5.21 – Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Клієнт виплачує гроші на рік, тоді до нього приходить спеціаліст із впровадження інформаційних систем і встановлює ПЗ на комп'ютер клієнта.	Встановити програмний продукт на комп'ютери клієнтів.	Один посередник – спеціаліст по впровадженню інформаційних систем.	Канал збуту одного рівня.

5.6 Висновки до розділу

В даному розділі було повністю виконано перший етап розроблення стартап-проекту, а саме, виконано маркетинговий аналіз стартап-проекту.

За допомогою нього можна сказати, що існує можливість ринкової комерціалізації проекту, адже на ринку наявний попит на системи автономного керування ТЗ, до того ж рентабельність роботи є досить високою.

З огляду на потенційну групу клієнтів, а саме, малий бізнес, що має вебсайт з приватним контентом для своїх клієнтів, та іноваційність технології є великі перспективи впровадження даного програмного забезпечення.

Для ринкової реалізації проекту доцільно обрати таку альтернативу впровадження: створення MVP.

ВИСНОВКИ

В ході проведеного аналізу були виявлені переваги і недоліки застосування нейронних мереж різних архітектур, з яких було обрано найбільш ефективна. Розроблена на основі CNN система виявлення і розпізнавання дорожніх знаків, дорожньої розмітки та ТЗ на зображенні володіє такими характеристиками, як висока шумоустойчівость, швидкість і точність розпізнавання при різних станах (знак частково пошкоджений), а також при різних умовах зйомки (зйомка йде при поганому освітленні).

У відповідності з поставленими завданнями були отримані наступні результати:

- були вивчені теоретичні основи CNN і перспективи їх застосування для розпізнавання образів на зображенні;

- виконано огляд існуючих згорткових мереж і вивчена можливість їх використання для решнія завдання розпізнавання образів;

- виконано огляд існуючих бібліотек і фреймворків реалізують алгоритми глибинного навчання;

- виконано пошук архітектури нейронної мережі для поставленого завдання;

- розроблено додаток, що демонструє принцип роботи системи;

- виконано тестування мережі і обгрунтована її ефективність.

Перспективи дослідження:

У майбутньому є можливість зробити з данної системи систему автономного керування ТЗ, так як всі потрібні передумови для цього є.

Також можливо розробити мобільний додаток, який буде отримувати зображення з камери та образу його оброблювати відповідно до описаної в роботі архітектури.

ПЕРЕЛІК ПОСИЛАНЬ

1. Дж.Ту, Р.Гонсалес. Принципы распознавания образов. М., Мир, 1978. 412 с.
2. Гонсалес Р., Вудс Р. Цифровая обработка изображений. М.:Техносфера, 2005. 1072 с.
3. Каллан Р. Основные концепции нейронных сетей The Essence of Neural Networks First Edition. 1-ше. «Вильямс», 2001. 288 с. ISBN 5-8459-0210-X. (рос.)
4. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel: Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation, Winter 1989. Vol 1(4):P. 541-551.
6. J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization” *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, Feb. 2012. URL: http://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf
7. N. Srivastava, G. E. Hinton et al., Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014. URL: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf
8. Hiai, Fumio; Lin, Minghua (February 2017). "On an eigenvalue inequality involving the Hadamard product".
9. S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, *arXiv preprint* arXiv:1502.03167, Feb. 2015. URL: https://arxiv.org/abs/1502.03167
10. A. G. Howard, Some improvements on deep convolutional neural network based image classification, *arXiv preprint* arXiv:1312.5402, Dec. 2013. URL: https://arxiv.org/abs/1312.5402 85
11. S. J. Nowlan and G. E. Hinton, Simplifying neural networks by soft weight-sharing, *Neural computation*, vol. 4, no. 4, pp. 473–493, 1992. URL: https://www.cs.toronto.edu/~hinton/absps/sunspots.pdf

12. T. Xiao, J. Zhang et al., Error-driven incremental learning in deep convolutional neural network for large-scale image classification, *in International Conference on Multimedia*, no. 22. ACM, 2014, pp. 177– 186.
13. J. Ortigosa-Hernández, I. Inza, and J. A. Lozano, Towards competitive classifiers for unbalanced classification problems: A study on the performance scores, *arXiv preprint* arXiv:1608.08984, Aug. 2016. URL: https://arxiv.org/abs/1608.08984
14. F. Chollet, "Keras," URL: https://github.com/fchollet/keras, 2015
15. M. Abadi, A. Agarwal et al., Tensorflow: Large-scale machine learning on heterogeneous distributed systems, *arXiv preprint* arXiv:1603.04467, Mar. 2016. URL: https://arxiv.org/abs/1603.04467
16. G. Huang, Z. Liu, and K. Q. Weinberger, Densely connected convolutional networks, arXiv preprint arXiv:1608.06993, Aug. 2016. URL: https://arxiv.org/abs/1608.06993v1
17. P. Sermanet and Y. LeCun, "Traffic sign recognition with multiscale convolutional networks," in International Joint Conference on Neural Networks (IJCNN), Jul. 2011, pp. 2809–2813. URL: http://ieeexplore.ieee.org/document/6033589/16

ДОДАТОК

Лістинги програм

Лістинг файлу train.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""Trains, evaluates and saves the TensorDetect model."""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import json
import logging
import os
import sys

import scipy as scp

# configure logging
if 'TV_IS_DEV' in os.environ and os.environ['TV_IS_DEV']:
    logging.basicConfig(format='%(asctime)s %(levelname)s %(message)s',
                        level=logging.INFO,
                        stream=sys.stdout)
else:
    logging.basicConfig(format='%(asctime)s %(levelname)s %(message)s',
                        level=logging.INFO,
                        stream=sys.stdout)

# https://github.com/tensorflow/tensorflow/issues/2034#issuecomment-220820070
import numpy as np
import tensorflow as tf

flags = tf.app.flags
FLAGS = flags.FLAGS

sys.path.insert(1, os.path.realpath('incl'))
```

```
import tensorvision.train as train
import tensorvision.utils as utils
import tensorvision.core as core

import tensorflow_fcn

import time

import random

flags.DEFINE_string('name', None,
                    'Append a name Tag to run.')

flags.DEFINE_string('project', None,
                    'Append a name Tag to run.')

flags.DEFINE_string('logdir', None,
                    'Append a name Tag to run.')

flags.DEFINE_string('hypes', None,
                    'File storing model parameters.')

tf.app.flags.DEFINE_boolean(
    'save', True, ('Whether to save the run. In case --nosave (default) '
                   'output will be saved to the folder TV_DIR_RUNS/debug, '
                   'hence it will get overwritten by further runs.'))



def _print_training_status(hypes, step, loss_values, start_time, lr):

    # Prepare printing
    duration = (time.time() - start_time) / int(utils.cfg.step_show)
    examples_per_sec = hypes['solver']['batch_size'] / duration
    sec_per_batch = float(duration)

    if len(loss_values.keys()) >= 2:
        info_str = ('Step {step}/{total_steps}: losses = ({loss_value1:.2f}, '
                    '{loss_value2:.2f});'
                    ' lr = ({lr_value1:.2e}, {lr_value2:.2e}); '
                    '({sec_per_batch:.3f} sec)')
        losses = loss_values.values()
```

```
        lrs = lr.values()
        logging.info(info_str.format(step=step,
                                     total_steps=hypes['solver']['max_steps'],
                                     loss_value1=losses[0],
                                     loss_value2=losses[1],
                                     lr_value1=lrs[0],
                                     lr_value2=lrs[1],
                                     sec_per_batch=sec_per_batch)
                     )
    else:
        assert(False)


def build_training_graph(hypes, queue, modules, first_iter):
    """
    Build the tensorflow graph out of the model files.

    Parameters
    ----------
    hypes : dict
        Hyperparameters
    queue: tf.queue
        Data Queue
    modules : tuple
        The modules load in utils.

    Returns
    -------
    tuple
        (q, train_op, loss, eval_lists) where
        q is a dict with keys 'train' and 'val' which includes queues,
        train_op is a tensorflow op,
        loss is a float,
        eval_lists is a dict with keys 'train' and 'val'
    """

    data_input = modules['input']
    encoder = modules['arch']
    objective = modules['objective']
    optimizer = modules['solver']
```

```
reuse = {True: False, False: True}[first_iter]

scope = tf.get_variable_scope()

with tf.variable_scope(scope, reuse=reuse):

    learning_rate = tf.placeholder(tf.float32)

    # Add Input Producers to the Graph
    with tf.name_scope("Inputs"):
        image, labels = data_input.inputs(hypes, queue, phase='train')

    # Run inference on the encoder network
    logits = encoder.inference(hypes, image, train=True)

# Build decoder on top of the logits
decoded_logits = objective.decoder(hypes, logits, train=True)

# Add to the Graph the Ops for loss calculation.
with tf.name_scope("Loss"):
    losses = objective.loss(hypes, decoded_logits,
                            labels)

# Add to the Graph the Ops that calculate and apply gradients.
with tf.name_scope("Optimizer"):
    global_step = tf.Variable(0, trainable=False)
    # Build training operation
    train_op = optimizer.training(hypes, losses,
                                  global_step, learning_rate)

with tf.name_scope("Evaluation"):
    # Add the Op to compare the logits to the labels during evaluation.
    eval_list = objective.evaluation(
        hypes, image, labels, decoded_logits, losses, global_step)

    summary_op = tf.summary.merge_all()

graph = {}
graph['losses'] = losses
graph['eval_list'] = eval_list
graph['summary_op'] = summary_op
```

```
    graph['train_op'] = train_op
    graph['global_step'] = global_step
    graph['learning_rate'] = learning_rate

    return graph


def run_united_training(meta_hypes, subhypes, submodules, subgraph, tv_sess,
                        start_step=0):

    """Run one iteration of training."""
    # Unpack operations for later use
    summary = tf.Summary()
    sess = tv_sess['sess']
    summary_writer = tv_sess['writer']

    solvers = {}
    for model in meta_hypes['models']:
        solvers[model] = submodules[model]['solver']

    display_iter = meta_hypes['logging']['display_iter']
    write_iter = meta_hypes['logging'].get('write_iter', 5*display_iter)
    eval_iter = meta_hypes['logging']['eval_iter']
    save_iter = meta_hypes['logging']['save_iter']
    image_iter = meta_hypes['logging'].get('image_iter', 5*save_iter)

    models = meta_hypes['model_list']
    num_models = len(models)

    py_smoothers = {}
    dict_smoothers = {}
    for model in models:
        py_smoothers[model] = train.MedianSmoother(5)
        dict_smoothers[model] = train.ExpoSmoother(0.95)

    n = 0

    eval_names = {}
    eval_ops = {}
    for model in models:
        names, ops = zip(*subgraph[model]['eval_list'])
```

```
    eval_names[model] = names
    eval_ops[model] = ops

weights = meta_hypes['selection']['weights']
aweights = np.array([sum(weights[:i+1]) for i in range(len(weights))])
# eval_names, eval_ops = zip(*tv_graph['eval_list'])
# Run the training Step
start_time = time.time()
for step in xrange(start_step, meta_hypes['solver']['max_steps']):

    # select on which model to run the training step
    # select model randomly?
    if not meta_hypes['selection']['random']:
        if not meta_hypes['selection']['use_weights']:
            # non-random selection
            model = models[step % num_models]
        else:
            # non-random, some models are selected multiple times
            select = np.argmax((aweights > step % aweights[-1]))
            model = models[select]
    else:
        # random selection. Use weights
        # to increase chance
        r = random.random()
        select = np.argmax((aweights > r))
        model = models[select]

    lr = solvers[model].get_learning_rate(subhypes[model], step)
    feed_dict = {subgraph[model]['learning_rate']: lr}

    sess.run([subgraph[model]['train_op']], feed_dict=feed_dict)

    # Write the summaries and print an overview fairly often.
    if step % display_iter == 0:
        # Print status to stdout.
        loss_values = {}
        eval_results = {}
        lrs = {}
        if select == 1:
            logging.info("Detection Loss was used.")
        else:
```

```
    logging.info("Segmentation Loss was used.")
for model in models:
    loss_values[model] = sess.run(subgraph[model]['losses']
                                  ['total_loss'])

    eval_results[model] = sess.run(eval_ops[model])
    dict_smoothers[model].update_weights(eval_results[model])
    lrs[model] = solvers[model].get_learning_rate(subhypes[model],
                                                  step)

_print_training_status(meta_hypes, step,
                       loss_values,
                       start_time, lrs)

for model in models:
    train._print_eval_dict(eval_names[model], eval_results[model],
                           prefix='   (raw)')

    smoothed_results = dict_smoothers[model].get_weights()

    train._print_eval_dict(eval_names[model], smoothed_results,
                           prefix='(smooth)')

output = sess.run(subgraph['debug_ops'].values())

for name, res in zip(subgraph['debug_ops'].keys(), output):
    logging.info("{} : {}".format(name, res))

if step % write_iter == 0:
    # write values to summary
    summary_str = sess.run(tv_sess['summary_op'],
                           feed_dict=feed_dict)
    summary_writer.add_summary(summary_str,
                               global_step=step)
    for model in models:
        summary.value.add(tag='training/%s/total_loss' % model,
                          simple_value=float(loss_values[model]))
        summary.value.add(tag='training/%s/learning_rate' % model,
                          simple_value=lrs[model])
    summary_writer.add_summary(summary, step)
    # Convert numpy types to simple types.
```

```
        if False:
            eval_results = np.array(eval_results)
            eval_results = eval_results.tolist()
            eval_dict = zip(eval_names[model], eval_results)
            train._write_eval_dict_to_summary(eval_dict,
                                              'Eval/%s/raw' % model,
                                              summary_writer, step)
            eval_dict = zip(eval_names[model], smoothed_results)
            train._write_eval_dict_to_summary(eval_dict,
                                              'Eval/%s/smooth' % model,
                                              summary_writer, step)

    # Reset timer
    start_time = time.time()

# Do a evaluation and print the current state
if (step) % eval_iter == 0 and step > 0 or \
   (step + 1) == meta_hypes['solver']['max_steps']:
    # write checkpoint to disk

    logging.info('Running Evaluation Scripts.')
    for model in models:
        eval_dict, images = submodules[model]['eval'].evaluate(
            subhypes[model], sess,
            subgraph[model]['image_pl'],
            subgraph[model]['inf_out'])

        train._write_images_to_summary(images, summary_writer, step)

        if images is not None and len(images) > 0:

            name = str(n % 10) + '_' + images[0][0]
            image_dir = subhypes[model]['dirs']['image_dir']
            image_file = os.path.join(image_dir, name)
            scp.misc.imsave(image_file, images[0][1])
            n = n + 1

        logging.info("%s Evaluation Finished. Results" % model)

        logging.info('Raw Results:')
        utils.print_eval_dict(eval_dict, prefix='(raw)   ')
```

```
            train._write_eval_dict_to_summary(
                eval_dict, 'Evaluation/%s/raw' % model,
                summary_writer, step)

            logging.info('Smooth Results:')
            names, res = zip(*eval_dict)
            smoothed = py_smoothers[model].update_weights(res)
            eval_dict = zip(names, smoothed)
            utils.print_eval_dict(eval_dict, prefix='(smooth)')
            train._write_eval_dict_to_summary(
                eval_dict, 'Evaluation/%s/smoothed' % model,
                summary_writer, step)

            if step % image_iter == 0 and step > 0 or \
                    (step + 1) == meta_hypes['solver']['max_steps']:
                train._write_images_to_disk(meta_hypes, images, step)
        logging.info("Evaluation Finished. All results will be saved to:")
        logging.info(subhypes[model]['dirs']['output_dir'])

        # Reset timer
        start_time = time.time()

    # Save a checkpoint periodically.
    if (step) % save_iter == 0 and step > 0 or \
       (step + 1) == meta_hypes['solver']['max_steps']:
        # write checkpoint to disk
        checkpoint_path = os.path.join(meta_hypes['dirs']['output_dir'],
                                       'model.ckpt')
        tv_sess['saver'].save(sess, checkpoint_path, global_step=step)
        # Reset timer
        start_time = time.time()
  return


def _recombine_2_losses(meta_hypes, subgraph, subhypes, submodules):
  if meta_hypes['loss_build']['recombine']:
    # Computing weight loss
    segmentation_loss = subgraph['segmentation']['losses']['xentropy']
    detection_loss = subgraph['detection']['losses']['loss']

    reg_loss_col = tf.GraphKeys.REGULARIZATION_LOSSES
```

```
    weight_loss = tf.add_n(tf.get_collection(reg_loss_col),
                           name='reg_loss')

    if meta_hypes['loss_build']['weighted']:
        w = meta_hypes['loss_build']['weights']
        total_loss = segmentation_loss*w[0] + \
            detection_loss*w[1] + weight_loss
        subgraph['segmentation']['losses']['total_loss'] = total_loss
    else:
        total_loss = segmentation_loss + detection_loss + weight_loss
        subgraph['segmentation']['losses']['total_loss'] = total_loss

    for model in meta_hypes['model_list']:
        hypes = subhypes[model]
        modules = submodules[model]
        optimizer = modules['solver']
        gs = subgraph[model]['global_step']
        losses = subgraph[model]['losses']
        lr = subgraph[model]['learning_rate']
        subgraph[model]['train_op'] = optimizer.training(hypes, losses,
                                                         gs, lr)


def _recombine_3_losses(meta_hypes, subgraph, subhypes, submodules):
    if meta_hypes['loss_build']['recombine']:
        # Read all losses
        segmentation_loss = subgraph['segmentation']['losses']['xentropy']
        detection_loss = subgraph['detection']['losses']['loss']
        road_loss = subgraph['road']['losses']['loss']

        reg_loss_col = tf.GraphKeys.REGULARIZATION_LOSSES

        weight_loss = tf.add_n(tf.get_collection(reg_loss_col),
                               name='reg_loss')

        # compute total loss
        if meta_hypes['loss_build']['weighted']:
            w = meta_hypes['loss_build']['weights']
            # use weights
            total_loss = segmentation_loss*w[0] + \
                detection_loss*w[1] + road_loss*w[2] + weight_loss
```

```
        else:
            total_loss = segmentation_loss + detection_loss + road_loss \
                + weight_loss

        # Build train_ops using the new losses
        subgraph['segmentation']['losses']['total_loss'] = total_loss
        for model in meta_hypes['models']:
            hypes = subhypes[model]
            modules = submodules[model]
            optimizer = modules['solver']
            gs = subgraph[model]['global_step']
            losses = subgraph[model]['losses']
            lr = subgraph[model]['learning_rate']
            subgraph[model]['train_op'] = optimizer.training(hypes, losses,
                                                gs, lr)


def load_united_model(logdir):
    subhypes = {}
    subgraph = {}
    submodules = {}
    subqueues = {}

    subgraph['debug_ops'] = {}

    first_iter = True

    meta_hypes = utils.load_hypes_from_logdir(logdir, subdir="",
                                      base_path='hypes')
    for model in meta_hypes['model_list']:
        subhypes[model] = utils.load_hypes_from_logdir(logdir, subdir=model)
        hypes = subhypes[model]
        hypes['dirs']['output_dir'] = meta_hypes['dirs']['output_dir']
        hypes['dirs']['image_dir'] = meta_hypes['dirs']['image_dir']
        hypes['dirs']['data_dir'] = meta_hypes['dirs']['data_dir']
        submodules[model] = utils.load_modules_from_logdir(logdir,
                                               dirname=model,
                                               postfix=model)

        modules = submodules[model]
```

```
    logging.info("Build %s computation Graph.", model)
    with tf.name_scope("Queues_%s" % model):
        subqueues[model] = modules['input'].create_queues(hypes, 'train')

    logging.info('Building Model: %s' % model)

    subgraph[model] = build_training_graph(hypes,
                                           subqueues[model],
                                           modules,
                                           first_iter)

    first_iter = False

if len(meta_hypes['model_list']) == 2:
    _recombine_2_losses(meta_hypes, subgraph, subhypes, submodules)
else:
    _recombine_3_losses(meta_hypes, subgraph, subhypes, submodules)

hypes = subhypes[meta_hypes['model_list'][0]]

tv_sess = core.start_tv_session(hypes)
sess = tv_sess['sess']
saver = tv_sess['saver']

cur_step = core.load_weights(logdir, sess, saver)
for model in meta_hypes['model_list']:
    hypes = subhypes[model]
    modules = submodules[model]
    optimizer = modules['solver']

    with tf.name_scope('Validation_%s' % model):
        tf.get_variable_scope().reuse_variables()
        image_pl = tf.placeholder(tf.float32)
        image = tf.expand_dims(image_pl, 0)
        inf_out = core.build_inference_graph(hypes, modules,
                                             image=image)
        subgraph[model]['image_pl'] = image_pl
        subgraph[model]['inf_out'] = inf_out

    # Start the data load
    modules['input'].start_enqueuing_threads(hypes, subqueues[model],
```

```
                                    'train', sess)

    target_file = os.path.join(meta_hypes['dirs']['output_dir'], 'hypes.json')
    with open(target_file, 'w') as outfile:
        json.dump(meta_hypes, outfile, indent=2, sort_keys=True)

    return meta_hypes, subhypes, submodules, subgraph, tv_sess, cur_step


def build_united_model(meta_hypes):

    logging.info("Initialize training folder")

    subhypes = {}
    subgraph = {}
    submodules = {}
    subqueues = {}

    subgraph['debug_ops'] = {}

    base_path = meta_hypes['dirs']['base_path']
    first_iter = True

    for model in meta_hypes['model_list']:
        subhypes_file = os.path.join(base_path, meta_hypes['models'][model])
        with open(subhypes_file, 'r') as f:
            logging.info("f: %s", f)
            subhypes[model] = json.load(f)

        hypes = subhypes[model]
        utils.set_dirs(hypes, subhypes_file)
        hypes['dirs']['output_dir'] = meta_hypes['dirs']['output_dir']
        hypes['dirs']['data_dir'] = meta_hypes['dirs']['data_dir']
        train.initialize_training_folder(hypes, files_dir=model,
                                         logging=first_iter)
        meta_hypes['dirs']['image_dir'] = hypes['dirs']['image_dir']
        submodules[model] = utils.load_modules_from_hypes(
            hypes, postfix="_%s" % model)
        modules = submodules[model]

        logging.info("Build %s computation Graph.", model)
```

```
    with tf.name_scope("Queues_%s" % model):
        subqueues[model] = modules['input'].create_queues(hypes, 'train')

    logging.info('Building Model: %s' % model)

    subgraph[model] = build_training_graph(hypes,
                                           subqueues[model],
                                           modules,
                                           first_iter)

    first_iter = False

if len(meta_hypes['models']) == 2:
    _recombine_2_losses(meta_hypes, subgraph, subhypes, submodules)
else:
    _recombine_3_losses(meta_hypes, subgraph, subhypes, submodules)

hypes = subhypes[meta_hypes['model_list'][0]]

tv_sess = core.start_tv_session(hypes)
sess = tv_sess['sess']
for model in meta_hypes['model_list']:
    hypes = subhypes[model]
    modules = submodules[model]
    optimizer = modules['solver']

    with tf.name_scope('Validation_%s' % model):
        tf.get_variable_scope().reuse_variables()
        image_pl = tf.placeholder(tf.float32)
        image = tf.expand_dims(image_pl, 0)
        inf_out = core.build_inference_graph(hypes, modules,
                                             image=image)
        subgraph[model]['image_pl'] = image_pl
        subgraph[model]['inf_out'] = inf_out

    # Start the data load
    modules['input'].start_enqueuing_threads(hypes, subqueues[model],
                                             'train', sess)

target_file = os.path.join(meta_hypes['dirs']['output_dir'], 'hypes.json')
with open(target_file, 'w') as outfile:
```

```
        json.dump(meta_hypes, outfile, indent=2, sort_keys=True)

    return subhypes, submodules, subgraph, tv_sess


def main(_):
    utils.set_gpus_to_use()

    load_weights = tf.app.flags.FLAGS.logdir is not None

    if not load_weights:
        with open(tf.app.flags.FLAGS.hypes, 'r') as f:
            logging.info("f: %s", f)
            hypes = json.load(f)
    utils.load_plugins()

    if 'TV_DIR_RUNS' in os.environ:
        os.environ['TV_DIR_RUNS'] = os.path.join(os.environ['TV_DIR_RUNS'],
                                                 'MultiNet')

    with tf.Session() as sess:

        if not load_weights:
            utils.set_dirs(hypes, tf.app.flags.FLAGS.hypes)
            utils._add_paths_to_sys(hypes)

            # Build united Model
            subhypes, submodules, subgraph, tv_sess = build_united_model(hypes)
            start_step = 0
        else:
            logdir = tf.app.flags.FLAGS.logdir
            logging_file = os.path.join(logdir, "output.log")
            utils.create_filewrite_handler(logging_file, mode='a')
            hypes, subhypes, submodules, subgraph, tv_sess, start_step = \
                load_united_model(logdir)
            if start_step is None:
                start_step = 0

        # Run united training
        run_united_training(hypes, subhypes, submodules, subgraph,
                            tv_sess, start_step=start_step)
```

```
        # stopping input Threads
        tv_sess['coord'].request_stop()
        tv_sess['coord'].join(tv_sess['threads'])


if __name__ == '__main__':
    tf.app.run()
```